AFRL-SN-RS-TR-2001-194
Final Technical Report
September 2001

HIGH PERFORMANCE COMPUTING SUPPORT FOR ADVANCED RADAR TECHNOLOGY RESEARCH CONSORTIUM

Maui High Performance Computing Center

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. D138

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-SN-RS-TR-2001-194 has been reviewed and is approved for publication.

APPROVED:

E. DOUGLAS LYNCH, JR.
Project Engineer

FOR THE DIRECTOR:

ROBERT G. POLCE
Chief, Rome Operations Office
Sensors Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/SNRT, 26 Electronic Pky, Rome, NY 13441-4514. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

HIGH PERFORMANCE COMPUTING SUPPORT FOR ADVANCED RADAR TECHNOLOGY RESEARCH CONSORTIUM

Donald J. Fabozzi, II
Blaise Barney
Joe Fogler
Mike Koligman
Steve Jackett
Brendan Bradley

Contractor: Maui High Performance Computing Center
Contract Number: F30602-95-C-0117
Effective Date of Contract: 13 June 1995
Contract Expiration Date: 31 January 2000
Short Title of Work: High Performance Computing Support for Advanced Radar Technology Research Consortium
Period of Work Covered: Jun 95 – Jan 00
Principal Investigator: Donald J. Fabozzi, II
Phone: (808) 879-5077
AFRL Project Engineer: E. Douglas Lynch, Jr.
Phone: (315) 330-4515

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Stephen A. Scott, AFRL/SNRT, 26 Electronic Pky, Rome, NY.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2001	Final Jun 95 - Jan 00

4. TITLE AND SUBTITLE

HIGH PERFORMANCE COMPUTING SUPPORT FOR ADVANCED RADAR TECHNOLOGY RESEARCH CONSORTIUM

5. FUNDING NUMBERS

C - F30602-95-C-0117
PE - 63226E
PR - MSEF
TA - 05
WU - 02

6. AUTHOR(S)

Donald J. Fabozzi, II, Blaise Barney, Joe Fogler, Mike Koligman, Steve Jackett, and Brendan Bradley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Maui High Performance Computing Center
550 Lipoa Parkway
Kihei HI 96753

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/SNRT
26 Electronic Pky
Rome NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-SN-RS-TR-2001-194

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: E. Douglas Lynch, Jr./SNRT/(315) 330-4515

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report encapsulated many provisions by the MHPCC to the Advanced Early Warning (AEW) community including compute time, online radar and IR databases, Khoros toolboxes, and MATLAB 1parallelization.

14. SUBJECT TERMS

Advanced Early Warning (AEW), RSTER, Matlab, Khoros

15. NUMBER OF PAGES

294

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Table Of Contents

List of Figures

Section 2.0

Section 6.0

Section 12.0

Section 13.0

Section 14.0

Section 15.0

Section 17.0

List of Tables

Section 14

1. Executive Summary

Executive Summary

The Maui High Performance Computing Center (MHPCC) is pleased to submit the final report for the contract "High Performance Computing Support For Advanced Radar Technology Research Consortium" for the period from June 29, 1995 to March 31, 1999. The project support included high performance computing cycles on the MHPCC Scalable Parallel (SP) system, software development in the areas of Khoros and MATLAB-based algorithms, and database development and management of the Radar Surveillance Technology Experimental Radar (RSTER) data and Airborne Infrared Measurement System (AIRMS). The project also provided hardware support, including MHPCC's High Performance Storage System (HPSS), dedicated program servers, disk, memory, and peripherals. Also installed to support this project were software packages such as Khoros 1.0P5 and V2.2 (Pro), MATLAB V4.2 and V5.0, Iona's Orbix 1.0 and 2.3, and IBM's Merchantile Solver. This report provides a complete description of each subtask, along with the entire library of generated papers, viewgraph presentations, and source code listings. The following paragraphs provide an overview of each of the contract requirements, followed by a table listing of each deliverable. This work was entirely supported by the Defense Advanced Research Projects Agency/ Sensor Technology Office (DARPA/STO) with guidance and assistance from Air Force Research Laboratory – Rome Site, Science Applications International Corp. (SAIC), and Massachusetts Institute of Technology/ Lincoln Laboratory (MIT/LL).

MHPCC established the Common Research Environment for STAP Technology (CREST) World Wide Web (WWW) server with direct access to the SP environment. This server is aliased to wwwcrest.mhpcc.edu and serves the program information via the WWW. This server provides access to the online distribution of the Rome Laboratory Space-Time Adaptive Processing/ Algorithm Development Tool (RLSTAP/ADT) and the CREST and AIRMS data libraries.

MHPCC integrated an administration system to assign, maintain, and track CREST researcher accounts. The account CPU hours were tabulated monthly and provided in the Quarterly Progress Reports.

MHPCC also provided dedicated help desk support for government approved researchers. A dedicated help desk system was set up to track, remedy, and catalog all assistance to approved researchers.

The **"Investigation of Parallelization of AEW Simulations Using the Rome Laboratory Space Time Adaptive Processing Algorithm Development Tool (RLSTAP/ADT)"** subtask investigated and implemented four technologies into Khoros' Cantata visual programming environment. Khoros is a third-party software product which includes over 250 mathematical, signal and image processing libraries along with the visual programming environment, Cantata. The technologies investigated include parallelization techniques with Khoros, integrating MATLAB algorithms into Khoros, implementing data compression utilities into Khoros, and conducting a thorough review of Rome Laboratory's RLSTAP/ADT operation under Khoros 2.1. Included in this

report are the viewgraphs from the September 30, 1997 Final Design review held at the University of New Mexico Galles Building. Also included are the manual pages for the parallel tools and compression utilities, and a copy of the paper "Khoros-based High Performance Computing Techniques for Radar Signal Processing" presented at the 1997 Khoros Symposium. The critical design reviews from each of the four subtasks along with the final report "RLSTAP Validation in Khoros 2.1" are also included. All works were performed by MHPCC and the Albuquerque High Performance Computing Center (AHPCC).

The "Investigation of Parallelization of Airborne Early Warning (AEW) Simulations Written in MATLAB" subtask details the results of the investigations of three approaches to high performance computing with MATLAB. The areas are in compiler and translator approaches, distributed MATLAB techniques, and conversion and integration techniques using Message Passing Interface (MPI) constructs. The approaches are investigated in the areas of speedup, scalability, portability, training time, and other miscellaneous advantages and disadvantages. This report includes a survey of the available techniques in MATLAB-based computing, followed by a deeper study of one candidate from each of the three categories. The packages evaluated were the Mathwork's MATLAB compiler, Cornell Theory Center's (CTC) MultiMATLAB, and Integrated Sensors Incorporated's (ISI) RTExpress. Also included are support utilities and user's guides for operation of MATLAB compiler and MultiMATLAB at the MHPCC.

The original Airborne Infrared Measurment System (AIRMS) program was a Defense Advanced Research Projects Agency/ Sensor Technology Office (DARPA/STO) sponsored effort which collected and processed 2 TBytes of data from a 24 inch infrared aperture, mounted on a Boeing 720B aircraft. MHPCC, along with Par Government Systems and Request Technologies, developed a server and interface for access of the AIRMS data product for online query, retrieval and processing. MHPCC utilized the technology from the DARPA Common Research Environment for STAP Technology (CREST) data library to distribute the AIRMS data through a World Wide Web (WWW) interface. In addition, the AIRMS data library project implemented the high performance IR data processing algorithms on the MHPCC SP system. Included in this report are the status of AIRMS flight database entries, the help file for the AIRMS Distributed Algorithm Stream (DAS) algorithm descriptions, execution instructions of the DAS at the MHPCC, and a technical paper presented at the SPIE '98 conference entitled "MHPCC Web-based IR Data Library and Simulation". The SPIE paper was produced on behalf of the AIRMS data library developers, Par Government Systems, at no additional cost to the contract. The AIRMS database was announced publicly on February 2, 1999 and is currently online at http://wwwcrest.mhpcc.edu/airms/query/.

MHPCC has been supplying System Services to support the IBM servers, disk, and software for the Advanced Signal Processing (ASP) program. Those include the IBM

server, sixteen Gigabytes of hard disk storage, and the software packages Orbix, MATLAB, and Khoros.

The **Northrop Grumman Dedicated Execution** project involved system and application support of Northrop Grumman who performed dedicated electromagnetic simulations on the MHPCC SP system. The SP configuration consisted of 4 frames of 8 Wide nodes (32 nodes) for electromagnetic airframe modeling. The MHPCC support for this execution included system problem resolution, system configuration, and data handling. The MHPCC also procured the "Merchantile Solver" which is an "out-of-core" linear solver developed by Ali Merchantile at IBM. This work began on July 21, 1997 and ended on January 6, 1998. For further information on this project, please contact the security office at MHPCC, (808)879-5077.

The **Rome Laboratory Dedicated Execution** operated on the same thirty-two SP "Wide" nodes as the Northrop Grumman project during the period of October 13-31, 1997. MHPCC support for this task included the areas of bulk parallelization, data handling, and dedicated support. For further information on this project please contact Mark Pugh at AFRL Rome Site, (315)330-7684.

The **University of Hawaii Parabolic Wave Equation (PWE)** project consisted of Lincoln Laboratory and the University of Hawaii performing a course-grain parallel implementation and execution of the Variable Terrain Radio Parabolic Equation (VTRPE) code. VTRPE was developed by Frank Ryan at the Naval Ocean Systems Center and is utilized by Lincoln Laboratory. The University of Hawaii implemented the parallelization of the VTRPE code and, with guidance and assistance from Lincoln Laboratory conducted runs on the MHPCC system. Enclosed is the final report of this work.

Two students from **Maui Community College** (MCC) provided support in the areas of web design, manual page development, and the parallel MATLAB task. The new web pages currently exist at http://wwwcrest.mhpcc.edu/. The manual page development task is documented in the report, "Enabling Online Help and Manual Pages for Khoros Pro 2.2". which details all of the utilities and configurations necessary for utilizing the Khoros manual pages on the IBM AIX 4.2 architecture. MCC also contributed to the report, "Using MultiMATLAB at MHPCC".

2. Khoros-Based High Performance Computing Techniques for Radar Signal Processing

D.J. Fabozzi
Maui High Performance Computing Center (MHPCC)
Ed Starczewski
AFRL Rome Lab
Tom Robbins
Steve Helmer
Kaman Sciences

14 March, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

"KHOROS-BASED HIGH PERFORMANCE COMPUTING TECHNIQUES FOR RADAR SIGNAL PROCESSING"

Presented at the Internation Khoros Symposium, March 24-26 1997

D.J. Fabozzi, Ed Starczewski, Tom Robbins, and Steve Helmer

The simulation of high fidelity radar surveillance data is very valuable for algorithm testing but analogously computationally expensive. The Khoros-based Rome Lab Space Time Adaptive Processing / Algorithm Development Tool (RLSTAP/ADT) simulates single degree coverage of terrain-specific or homogenous radar data but was designed for execution on the desktop workstation. To respond to the need for the simulation and processing of high fidelity Advanced Early Warning (AEW) radar systems, enhancements were implemented to the RLSTAP-Cantata structure which allow the efficient simulation of the full 360 degree surveillance volume on multiple processors of the distributed memory Maui High Performance Computer (MHPCC). Through a batch queuing technique and the integration of an MPI-based clutter generator, simulation and processing execution times are significantly reduced. This report details how these techniques are providing remote researchers with a powerful, user-friendly, and low-bandwidth interface to remote computing.

INTRODUCTION

In support of the DARPA/STO Advanced Signal Processing (ASP) Program, Rome Laboratory (RL) developed the "Rome Laboratory Space Time Adaptive Processing (STAP) Algorithm Development Tool (RLSTAP/ADT) to support a wide variety of Advanced Early Warning (AEW) experiments. Developed under the Khoros 1.0P5 environment, RLSTAP's primary technologies include the evaluation of measured radar data, the simulation of ground-based or airborne multi-channel data, jammers, clutter, and the development of new STAP algorithms.

Originally implemented on the desktop workstation, it was soon desired to utilize RLSTAP to investigate commercial radar technologies which required High Performance Computing (HPC). Upon selecting the Maui High Performance Computing Center (MHPCC) as the compute engine, the challenge then became how to embrace the *remote* compute power while maintaining the Khoros-Cantata user-friendly environment. With a low bandwidth connection to the MHPCC as the dominant operational constraint, the optimal architecture involved minimizing the communication by distributing data and precompiled libraries remotely at the MHPCC. This architecture further involved augmenting remote "client" versions of Khoros-Cantata to have the capability to record, transport, and execute experiments remotely. This implementation has proven very successful in that remote researchers can execute intensive simulations, generate United States Geological Survey (USGS) maps, and perform large data set quick look and signal processing experiments through this flexible, intuitive environment.

In fact, since its announcement in January, 1996, RLSTAP has been utilized by over 70 Department of Defense (DOD) organizations and contractors across the United States. As users return feedback into the development, the RLSTAP utility is continually evolving, primarily in the areas of High Performance Computing and the migration to Khoros 2. This report snapshots the current version of RLSTAP and details the Remote-RLSTAP augmentation, followed by an overview of the integration of a Message Passing Interface (MPI) version RLSTAP's most exhaustive component, monostatic clutter generation.

THE ROME LABORATORY SPACE-TIME ADAPTIVE PROCESSING ALGORITHM DEVELOPMENT TOOL (RLSTAP/ADT)

As mentioned, RLSTAP is a full end -to-end simulation and processing toolbox consisting of the following sub-toolboxes: Physical Model, Signal Processing, Receiver, Mathtools, Diagnostics, Recorded Data, Detection, Waveforms and Filters, and Remote Processing. As illustrated in Figure 1, each sub-tool box is accessible from the Cantata main form:

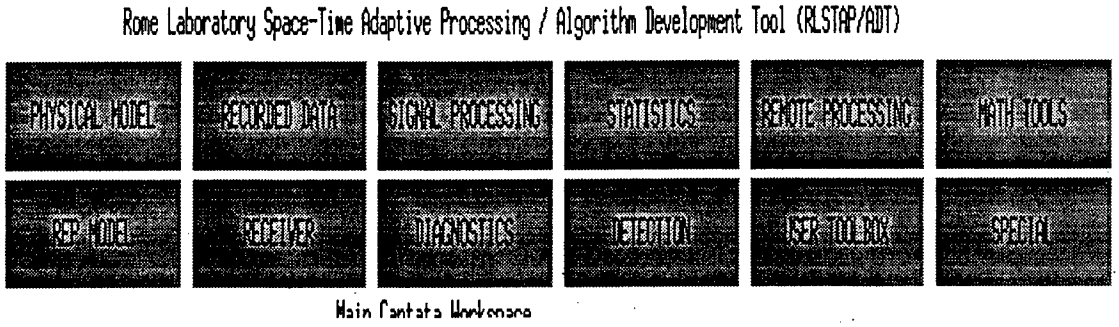


Figure 1

The Physical Model performs radar modeling and is divided into separate modules for the transmitter, transmitter platform, waveform, transmit antenna, receiver, receiver platform, and receiver antenna. The Physical Model (PM) also allows the analyst to generate realistic target, jammer, and clutter signals. The clutter generator simulates homogenous or user-specified locations using terrain height and terrain cover

information available in the USGS database to derive the line-of-sight visibility, grazing angle, and clutter type for each range-angle cell in a surveillance volume. The PM also:

- models Targets as point sources with user specified Radar Cross Section (RCS), range, angle, heading and velocity
- models ground-based or airborne barrage noise jammers with user specifiable location, orientation, radiated power, modulation type, center frequency, bandwidth, period, duty factor and sweep rate.

The Receiver Toolbox simulates the radar receiver through user-defined gains/losses, RMS gain variations, system noise figure, and system noise temperature.

The Signal Processing toolbox provides a library of radar signal processing functions such as Pulse Compression, Motion Compensation, Moving Target Indication, Doppler Subbanding, Beamforming, Steering Vector, STAP Rules, Covariance matrix, Diagonal Loading, Inverse Covariance matrix, and Adaptive Weights.

The Diagnostic Tools include XPRISM plotting, Animation, and various test facilities.

The Remote Processing Toolbox contains all of the facilities necessary to generate, transport, process, and visualize High Performance Computing experiments, as described in the following sections.

THE REQUIREMENTS FOR HIGH PERFORMANCE COMPUTING

The original version of RLSTAP performs static simulation/processing on one scan angle or Coherent Processing Interval (CPI) at a time. Recently, however, it has been required to rapidly simulate/process entire surveillance volumes to closer model active commercial radar systems.

Similarly, there often exists the need to efficiently examine large quantities of collected mission data. Data collection experiments occur continuously in the radar community and it is often the burden of the collecting organization to review the data quality and features. Automated processing systems are very valuable in these cases to fully review the complete data storage.

From a data simulation perspective, typical STAP experiments are performed on one look angle at a time and usually complete on the order of minutes. For example, the data generation and processing experiment in Figure 2 is a typical processing line-up,.

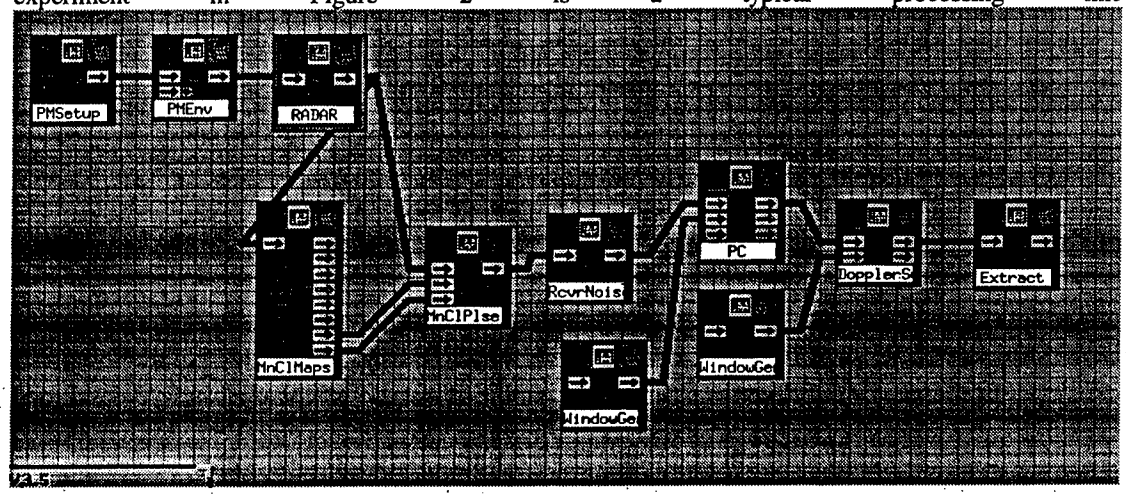


Figure 2

with the following specifications:

•number of range cells: (N_RCVR_RNGPTS)=112

•number of azimuth points in a range-angle map (N_RCVR_RA_MAP_AZPTS)=40

•number of range points in a range-angle map (N_RA_MAP_RNGPTS)=824

•number of pulses (N_PULSES)=256

•number of channels (N_ELEM)=26

The execution times were measured for various platforms, as illustrated in Figure 3:

COMPARITIVE EXECUTION TIMES FOR SIMULATION OF SINGLE DWELL, SERIAL EXECUTION

{CLUTTER REPRESENTS GREATEST PROCESSING LOAD}

PROCESSOR (hostname)	SPECS: CLOCK/MEM/CACHE	EXECUTION TIME (HR:MIN:SEC)	EXEC. TIME MnCIPlse (HR:MIN:SEC)	% TIME SPENT IN MnCIPlse	ESTIMATED COMPUTATION TIME, FULL LINEUP, 36 DWELLS(HR:MIN)	ESTIMATED COMPUTATION TIME, FULL LINEUP, 360 DWELLS(HR:MIN)
SPARC 2 (moses.oc.rl.af.mil)	30Mhz/32MB/none	REAL: 5:15.03 USER: 5:09.01 SYS: 87.9	REAL: 5:07.02 USER: 5:02.49 SYS: 65.6	97.4 97.9 74.5	REAL=189:06	REAL=1891:05
SPARC10 (mountain.oc.rl.af.mil)	Model: 41 40Mhz/128MB primary: 20KBI+16KBD secondary: none	2:17.30 2:13.50 .48	2:12.58 2:10.45 .34	96.1 97.3 73.0	82:30	825:20
SPARC 20 (crest.oc.rl.af.mil)	Model: 50 50Mhz/128MB primary: 20KBI+16KBD secondary: none	1:50.00 1.38.14 .29	1:47.21 1:35.58 .19	97.6 97.6 67.0	66:00	660:00
SPARC ULTRA (cordillera.oc.rl.af.mil)	Model: 170 167Mz/64MB primary: 16KBI+16KBD secondary: 512KB(I+D)	:49.15 :44.01 .11	:44.03 :42.52 .05	89.4 97.4 53.2	29:40	295:30
IBM RS6000 POWER PC60 (everest.ocsa.rl.af.mil)	Model: 41T 80Mhz/64MB primary:32KB(I+D) secondary: 0.5MB	:48.58 :41.45 .18	:46.27 :39.37 .15	95.0 94.0 80.0	29:20	293:40
IBM SP2 (tr2n16.mhpcc.edu)	POWER processor 66.5Mz/64MB primary: 31KBI+64KBD secondary: 0-1MB	:35.09 :22.33 .28	:32.58 :20.31 .25	93.0 92.1 87.5	21:01	210:55

REAL: Total wall clock time to load, execute, exit
USER: time for CPU to execute
SYS: OS time to service requests

Figure 3

As indicated in columns 6 and 7, full 360 degree simulations produce run times which are measured in days. Also, because the largest percentage of the overall processing is devoted to the generation of the clutter responses (as indicated in column 5) clutter generation itself also is an excellent candidate for High Performance Computing.

RLSTAP HIGH PERFORMANCE COMPUTING

The utilization of RLSTAP with the Maui High Performance Computing Center to solve these problems has taken two primary endeavors:

- A batch processing system to transport, execute, and visualize Cantata experiments between the remote researcher and the MHPCC and
- a Message Passing Interface (MPI) version of clutter generation.

RLSTAP BATCH PROCESSING WITH KHOROS 1

The basic architecture of Remote-RLSTAP consists of:

- Distributing the RLSTAP precompiled binaries to the user's local workstation and a "server" version at the MHPCC
- Locating any collected data is located at the MHPCC
- The interfaces for the "client" versions of RLSTAP which provide:
    - the offloading of single experiments to single MHPCC SP2-nodes
    - the generation of multiple experiments for execution on SP2 nodes
    - a mechanism to iterate over multiple parameters with the experiments
    - the review, download, and visualization of the results

The novelty of the Remote operation begins with the operation of "client" batch queuing interface. To capture the workspace action, a global "mode" glyph controls the interception of glyph invocations from the Cantata scheduler and determines whether to operate in *local* or *batch* mode. The execution path environment variable (*PATH*) specifies a leading directory (*rbin*) that contains shell scripts for each RLSTAP routine which contain control logic to select between the different modes. Thus, upon glyph execution, each rbin script decides, based on the environment files set by the "mode" glyph, whether to execute locally or to concatenate the calling glyph's output to the batch file

Next, batch processing filters and generators were implemented to allow the generation of multiple experiments with the capacity of varying the type and dimension of the processing. The batch files are then transported via automated FTP sessions to user's HOME directory at the MHPCC. There, a daemon assigns each job to a node on a first come, first serve basis. As illustrated in Figure 4, this Single Program Multiple Data (SPMD) type of programming uses a job mapping of one job per node:

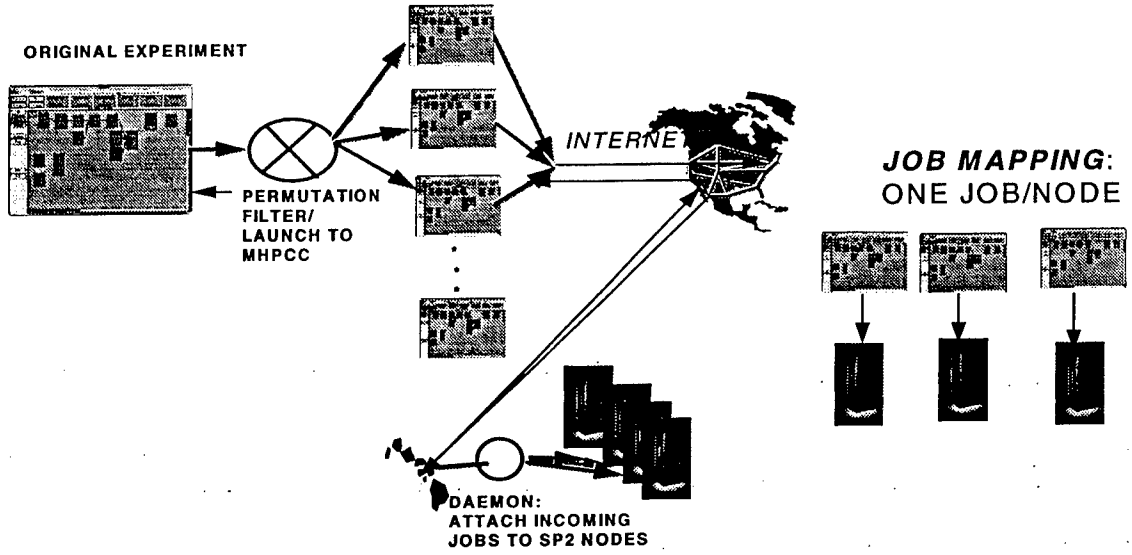


Figure 4

This environment was first utilized for the quick-look and STAP processing of the DARPA/STO Mountain Top data at the MHPCC. Since there are over 4000 data sets available and each contains between 10-100 CPI's per set, examination of the full library or inter-CPI features quickly becomes a challenge. Remote RLSTAP provided a solution to this problem whereby users can quickly batch off

multiple experiments, each to perform on separate CPIs. Figure 5 illustrates a frame sequence of range-doppler plots from Khoros' *"Animate"* that resulted from a typical multi-CPI batch experiment. This experiment answered the common question of "in which CPIs do the targets actually appear".

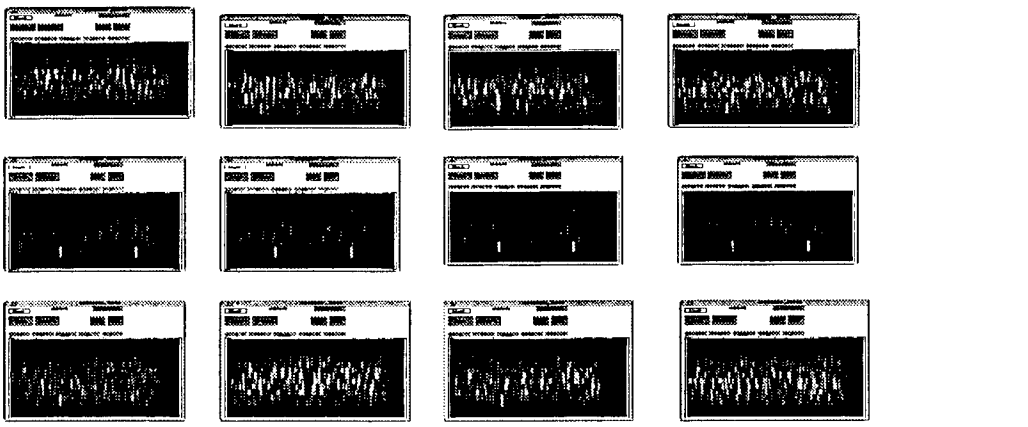


Figure 5

The MHPCC-Remote RLSTAP capabilities also provide:

- Signal and STAP processing over multi-CPI Mountain Top Data with the following degrees of freedom:

  * CPI #, diagonal loading, azimuth-elevation steering angle, STAP-center bin #, guard bin # , doppler subbanding

- Dynamic physical model multi-CPI radar scenario, transmit-receiver positions, target positions

- USGS mapping. Upon specification of a position and a range extent, the utility can generate and display any selected region in the United States. Though mapping is commonly available on WWW sites, this environment further provides the data files necessary to perform clutter simulation.

RLSTAP BATCH PROCESSING WITH KHOROS 2

The candidate design of the Khoros 2-based RLSTAP batch processing capability will be based on processing the standard Khoros workspace file. The workspace file is translated to batch processing scripts whereby the *cantata* programming constructs (loops, conditionals, variables and procedures) are supported in addition to a RLSTAP batch iteration mechanism.

DISTRIBUTED CLUTTER GENERATION

Though the batch processing system can perform off-line generations of multi-CPI surveillance scenarios, the computational workload to perform them is still great. As indicated in column 5 of Figure 3 this is primarily due to the generation of simulated clutter. Upon examination of the clutter algorithm, or "MnClPlse", it was quickly determined that the most significant improvement in execution would result from distributing the workload among multiple processors..

Prior to discussing the parallelization effort, however, a discussion of the clutter generation algorithm will provide an appreciation of the extent of processing involved. Since, there could potentially be a variety of factors contributing to the slow execution of an algorithm, the source code was examined to find the execution "hot spots". As illustrated in the simplified pseudocode version of the clutter generation in Figure 6, the bulk of the workload is in the "loop" section. To confirm this, timing calls were used to measure execution times as the data dimension increases, as indicated in Figure 7.

PRE-LOOP

```
"input op. parameters, choose fft size, allocate memory, set range seed"
ci_get_cltr_vars();
ci_get_cltr_vars3();
"pick fft size();
wrk*=kmalloc();
set_mg_seed();
"find location, value of range-azimuth map peak"
rl_get_cltr_ra_map_pk()
"generate clutter data cube"
rl_gen_cltr_rpe_map2():
```

LOOP

```
DO IELEM=1, N_ELEMENTS
    DO IPULSE=1, N_PULSES
        DO IANG=1,N_RCVR_RA_MAP_AZPTS
        "calculate spatial phase"
      Z_ACS_DG=AZ_ARRAY_PCS(IANG)-MOD360(AZ_PCS...)
        "extract range strobe from range-azimuth map"
            DO IR=1, N_RA_MAP_RNGPTS
            "calculate phase terms and add to clutter"
        V_CLTR_SUM(IR)=V_CLTR_SUM(IR)+V_CLTR(IR)*
        CMPLX(COS(PHI_SPATIAL+PHI_TEMPORAL),
        SIN(PHI_SPATIAL +PHI_TEMPORAL))
            END DO
        END DO
        "convolve sequence with transmit pulse"
        CONVOLVE(V_CLTR_SUM,TX_PULSE_WF,...)
    END DO
```

POST-LOOP

```
END DO
"write output cube"
rl_writeiq();
```

Figure 6

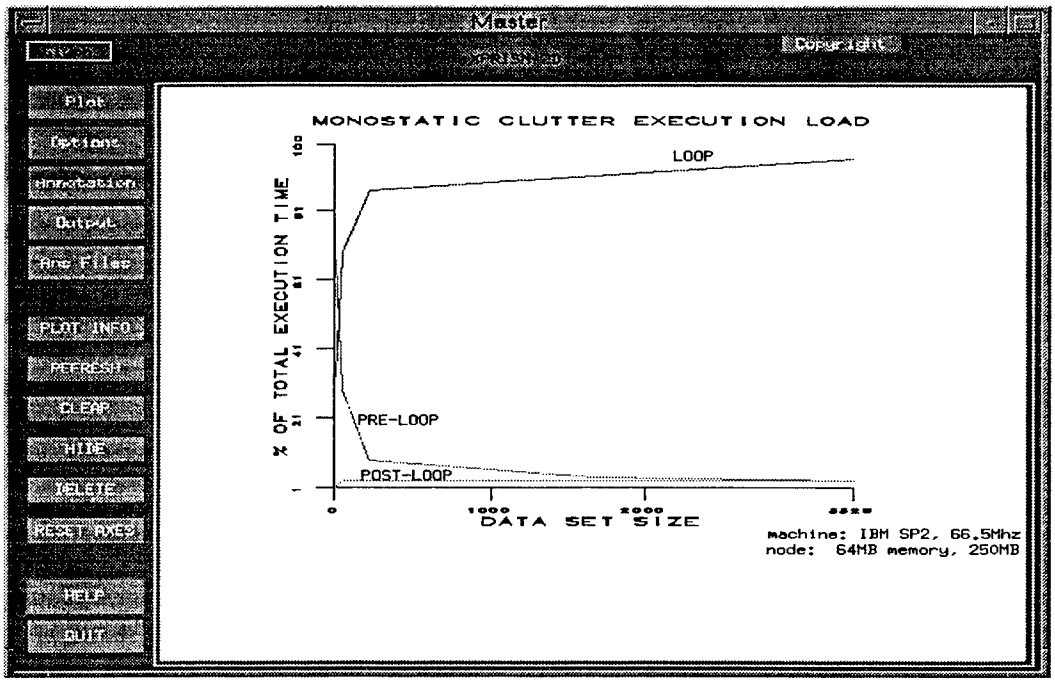


Figure 7

The algorithm is timely primarily because of the quantity of operations being performed. The entire surveillance volume is examined for the computation of each pulse-element point which is calculated using the standard radar range equation for clutter. The clutter simulation treats each range-angle cell as an individual point scatterer whose signal strength at the receiver is a function of the backscatter coefficient, range, atmospheric attenuation, antenna gain, and system gains/losses. The algorithm generates weighted sum of all the angle cells in a range ring and then convolves the sequence with the transmit pulse. The result is a sequence that is the sum of time delayed values from the all of the individual point scatterers illuminated by the transmit pulse.

To investigate the performance of this complex algorithm on the MHPCC SP2 processor, the serial version of MnClPlse was timed varying pulse and channel dimensions. As Figure 8 illustrates, the User time follows the System time closely, confirming that the data sizes used in the forthcoming evaluations will not cause delays in system activity such as paging, swapping, or system calls.

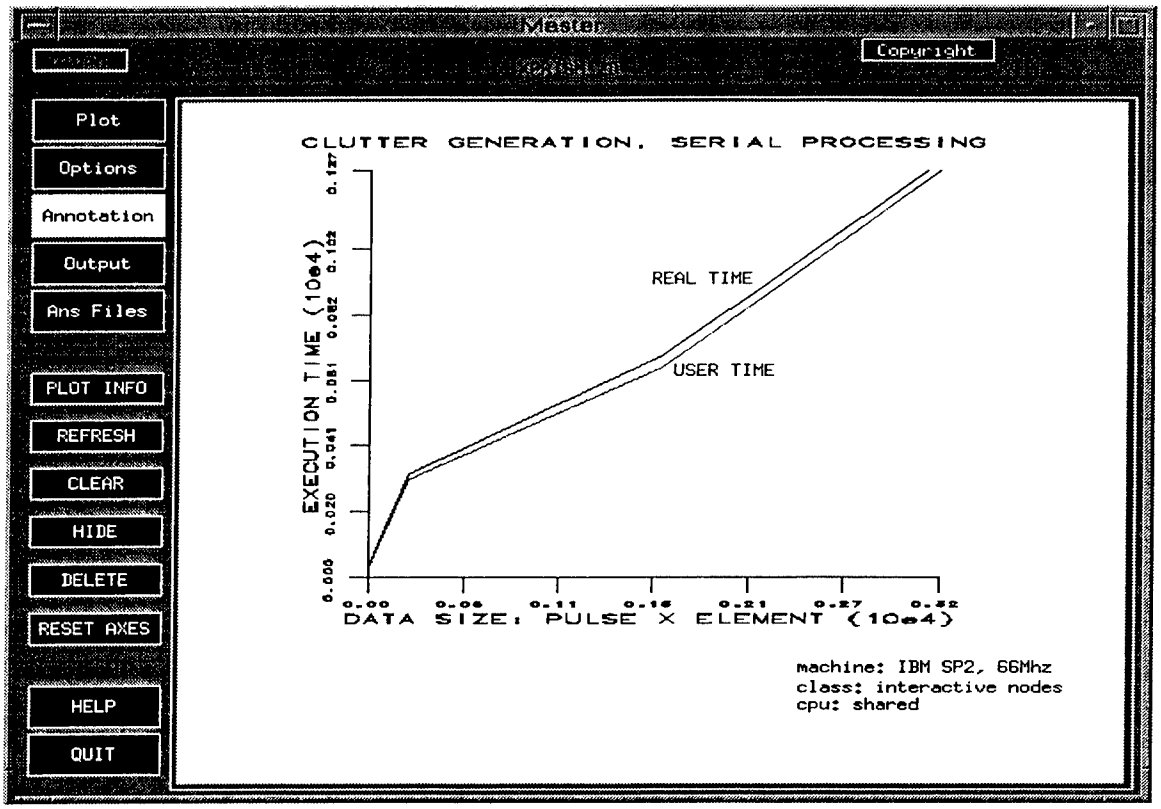


Figure 8

BATCH CLUTTER

Two distributed processing techniques were utilized to parallelize MnClPlse and the first utilized the remote processing "batch" technique discussed in the previous section. This technique consisted of generating multiple calls to the original serial version of MnClPlse, with each to generate a clutter "slice", rather than the entire RangePulseElement (RPE) cube, as illustrated in Figure 9. Upon completion, the RLSTAP client would then launch another process to combine the resultant element slices to form the RPE cube. Though this technique reduced computation time, the technique was not widely accepted due to its awkward manual operation.

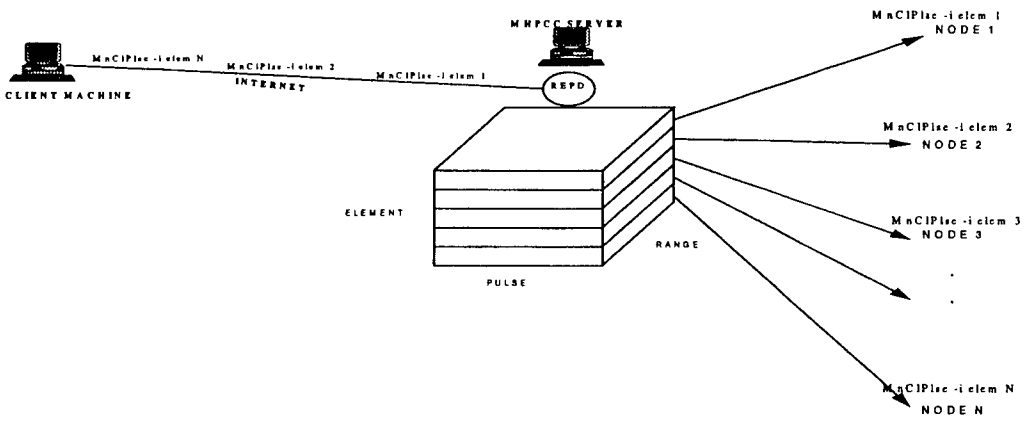


Figure 9

MPI CLUTTER

The Message Passing Interface version of MnClPlse also partitioned work up in the element dimension, but in a more robust fashion. MPI-Clutter allows RLSTAP clients to substitute the MnClPlse glyph with the MPI version (MnClMPI) to be invoked on the SP2, as illustrated in Figure 10.

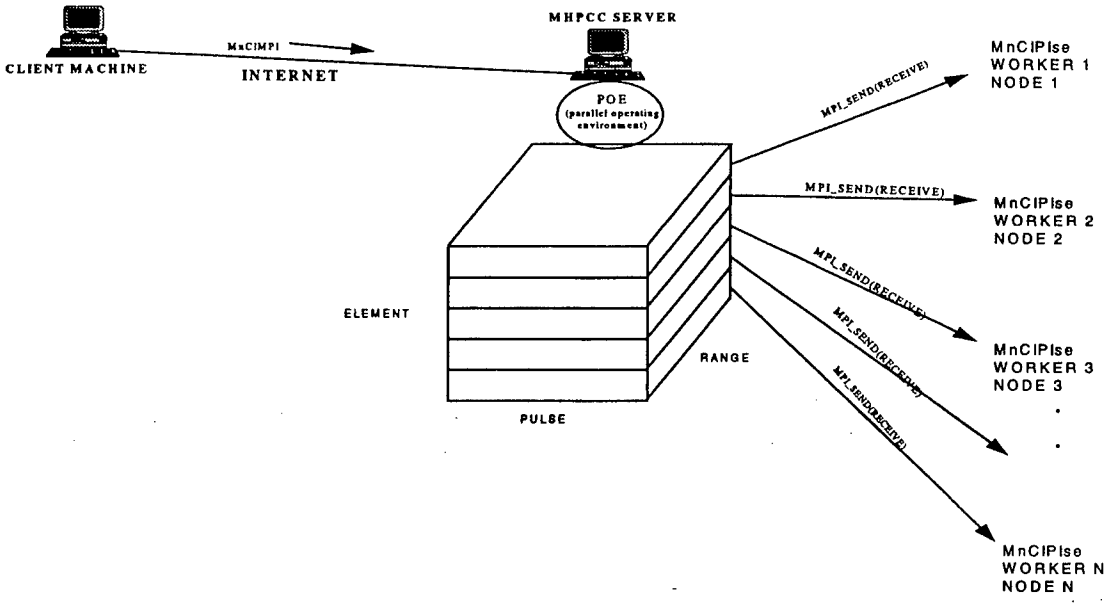


Figure 10

MPI CLUTTER INTEGRATION

The incorporation of message passing into MnClPlse was fairly straightforward. As mentioned, the division of labor is performed in the outer "element" do loop and, given that the number of elements may not always equal the number of processors, the algorithm was implemented in the following manner:

*** master ***

DO ielem=1, n_channels

if (ielem .le. numtasks)

"set up element index" MPI_IRECV((C_RPE(1,1,ELEM)),...

if (ielem .eq. ielem_stop)

"send out random numbers"

MPI_BCAST(C_RA_MAP()

if (ielem .gt. numtasks)

"wait for open node"

MPI_WAITANY()

"set up element index"

MPI_IRECV()

END DO

MPI_WAITANY

*** worker***

"receive random numbers"

MPI_BCAST(RA_MAP)

"generate clutter RP slice"

"send the processed channel to master"

MPI_ISEND()

MPI_WAIT

"receive next channel"

MPI_RECV()

THE TESTING OF MPI CLUTTER

To establish an optimal execution scenario, MnClMPI was evaluated by varying three characteristics: the communication system, the number of compute nodes, and the data dimensions. The curves labeled with communication system: "slow" indicate :

- the use of the *Internet Protocol* (IP) communication between nodes connected via the *ethernet*

Or "fast" as:

- use of the *User Space Protocol* (US) communication between nodes connected via the *high performance switch*

The following plots are measurements from clutter simulation of the following dimension:

- number of range cells: (N_RCVR_RNGPTS)=790
- number of azimuth points in a range-angle map (N_RCVR_RA_MAP_AZPTS)=256

•number of range points in a range-angle map (N_RA_MAP_RNGPTS)=695

•number of pulses (N_PULSES)=256

•number of channels (N_ELEM)=26

The plot in Figure 11 indicates that for relatively small data sizes PulseElement(PE)=(7,7), speedup due to parallelism is minimal.

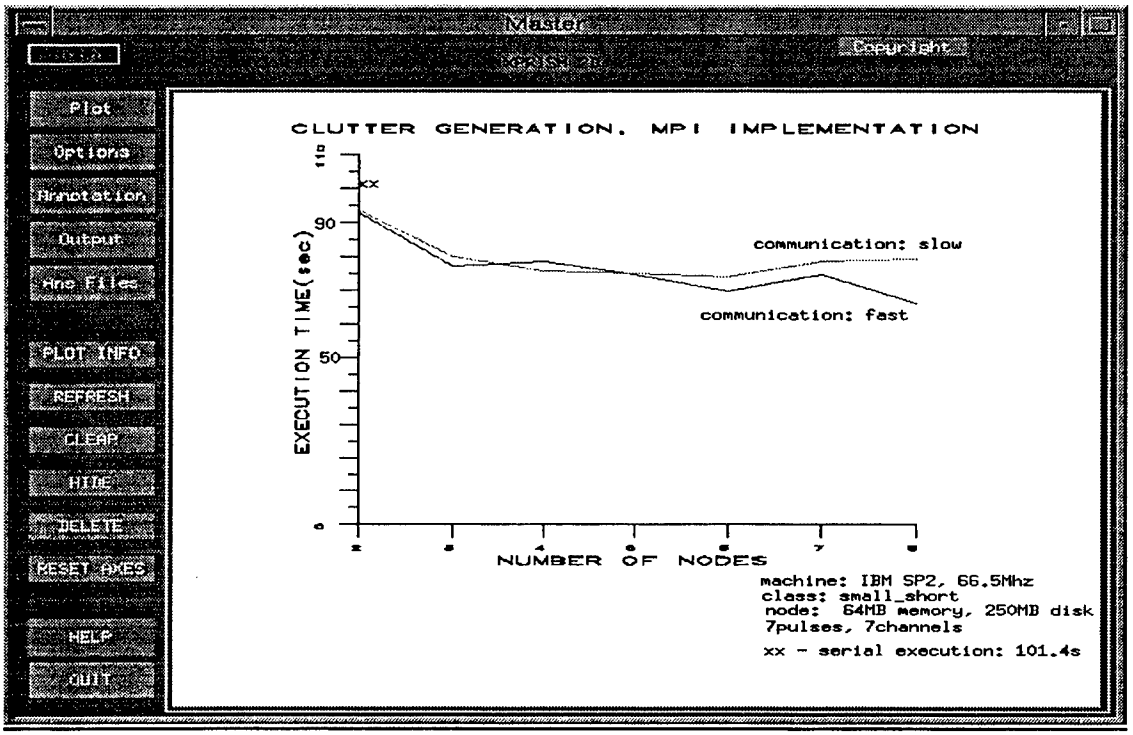


Figure 11

Figure 12, however, indicates a greater speedup, however from 320sec (serial) to about 100sec (parallel). Unfortunately performance improvement is almost fully realized at about 30-50% of the nodes available, a limitation we found throughout the testing of MnCIMPI. A possible explanation for this may be NFS-based disk contention. The plot also reflects that increased number of nodes further distinguishes communication performance.

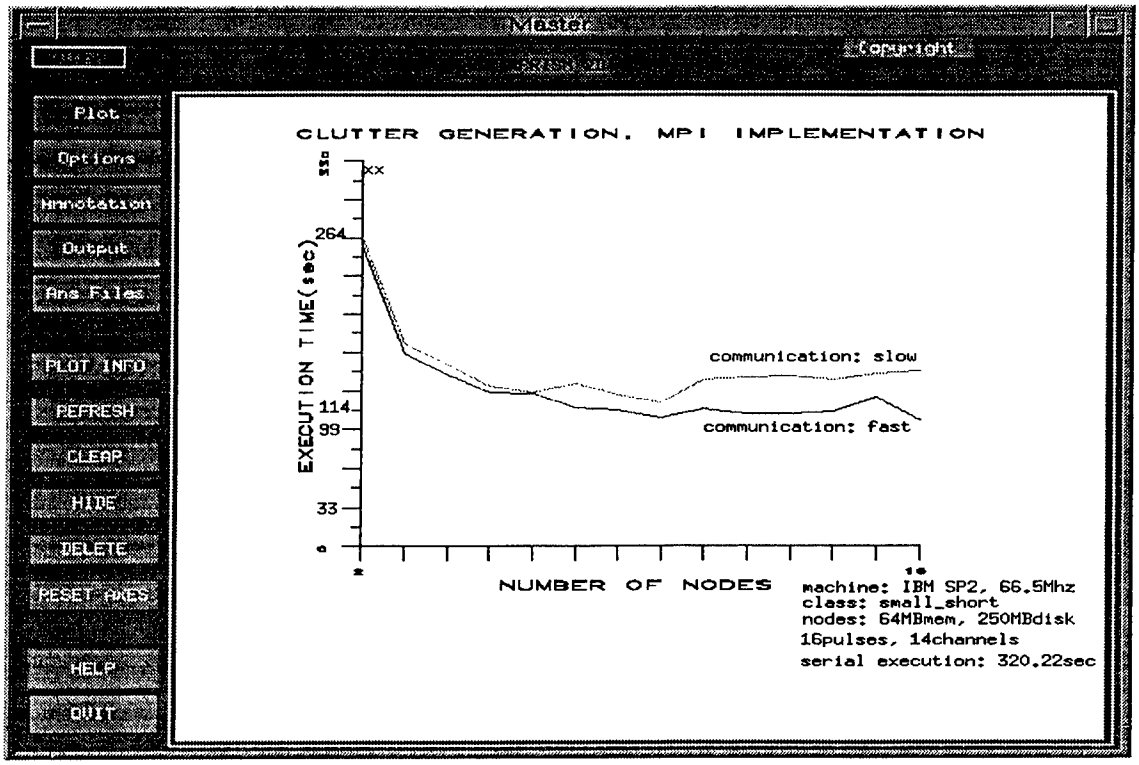


Figure 12

Figure 13 illustrates a greater improvement for larger data dimensions, for PE=(64,26). The serial version was timed at 690 seconds while the parallel version took about 250 seconds. Again, as the number of nodes increased, the *fast* communication system outperformed the *slow* system.

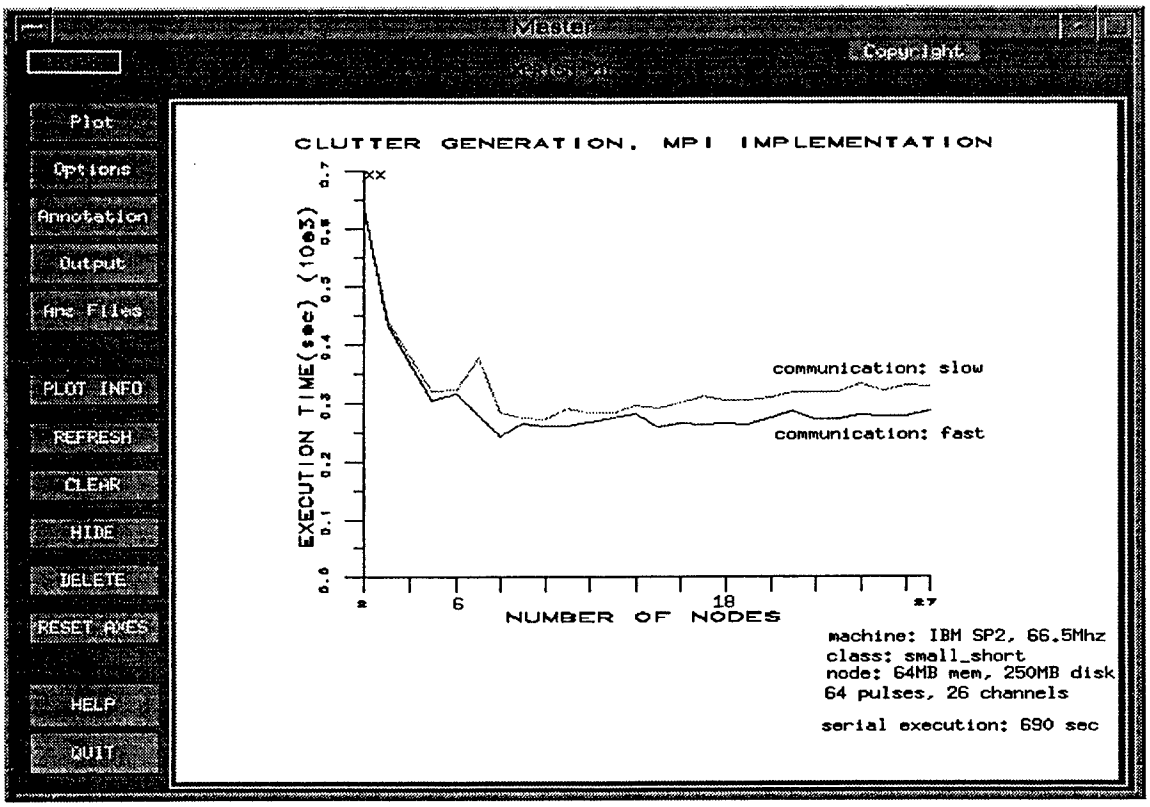


Figure 13

This MPI-based version provides numerous advantages over both the serial version and the "batch clutter" version:

- performance, minimal data transport - less internet traffic, data transfer is performed in compiled code
- user interface- user makes only one call to the clutter generation routine, data recombination is handled with the MPI constructs

FULL SURVEILLANCE VOLUME SIMULATION

The real question is "What kind of performance improvement can be achieved for a 360degree simulation?" The chart in Figure 14 indicates that for data dimensions of RPE=(790,256,26), the serial simulation has a duration of 2 hours, 21 minutes, while the parallel version performs in 16 minutes! Since clutter generation represents over 90% of our simulation experiment, we've linearly scaled the measurements to the full surveillance volume, (in 10degree increments). As indicated in the last column our 84 hour serial run executes in 9 hours running parallel

COMPARITIVE EXECUTION TIMES FOR CLUTTER SIMULATION
BOTH SERIAL AND PARALLEL EXECUTION

		10 NODES	27 NODES	ESTIMATED COMPUTATION TIME 36 DWELLS (HR:MIN)
SERIAL	2:21.00	XXXXX	XXXXX	84:36.00
PARALLEL: SLOW COMM:	THIN NODE	29.20	30.36	
	WIDE NODE	25.30	18.13	10:54
FAST COMM-1	THIN NODE	25.13	16.37	
	WIDE NODE	24.39	16.26	10:02
FAST COMM-2	THIN NODE	25.24	16.39	
	WIDE NODE	24.9	16.15	9:45

Figure 14

SUMMARY

The convergence of Khoros-based utilities and High Performance Computing is a natural one with the power and flexibility bestowed in each and there are countless cooperations between them to be realized. Remote RLSTAP and MPI Clutter are two offspring from this union which were implemented to address Advance Early Warning problems. Through this remote batch processing system and the high-speed clutter generation, remote users can now effectively simulate/process large surveillance volumes of high fidelity radar data through a user-friendly environment.

In the growing world of expensive commercial client-server development environments this project is also a refreshing example of the utilization of available resources to attack known DOD problems. The publicly available Khoros was selected as the parent software environment for RLSTAP, not only for economical reasons, but because of the novel user interface, extensibility, and large user base. These features facilitated this project extremely in the areas of graphics, development of the batch processing model, distributed data organization, and in the distribution of the RLSTAP "clients".

3. HPC for Advanced Early Warning Simulations in RLSTAP

Program Overview

Joe Fogler
Albuquerque High Performance Computing Center (AHPCC)

30 September, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

Task Areas

- Data compression (Brendan Bradley)
- Integration of STAP Matlab scripts (LLSTAP) into RLSTAP (Steven Jackett)
- Validation of RLSTAP K2 (Khoros 2.1) (Martha Ennis)
- Parallelization tools (Ruth Klundt and Ken Summers)

Data Compression

Motivation

- Air Early Warning (AEW) radar algorithm development and testing requires very large amounts of data
- Collaborative development may involve researchers in diverse geographic locations who need to exchange information

Objectives

- Add lossless data compression capabilities to the RLSTAP software environment for
  - RSTER-like radar data
  - Formed SAR images
- Utilize non-proprietary techniques

Data Compression, continued

Approach

- Conduct survey of lossless compression techniques
- Examine statistical properties of representative data
- Identify handful of most promising techniques
- Acquire or write standalone C-language data compression codes
- Test data compression codes on available data
- Downselect to best algorithms for RSTER and SAR data
- Write Khoros-compatible software tools (kroutines) for the RLSTAP environment

Integration of LLSTAP into RLSTAP

Motivation

- There are two predominant software development environments for collaborative algorithm development and experimentation - Khoros and Matlab
- Little interoperability between these two environments

Objectives

- Provide access to STAP algorithms written in Matlab script from within Khoros environment
- Investigate porting of scripts to C-language
- Develop an interface for calling scripts directly from Khoros without code conversion

Integration of LLSTAP into RLSTAP, continued

Code conversion approach

- Determine code hierarchy, dependencies, etc.
- Identify Matlab scripts that represent functions already implemented by existing Khoros toolboxes
- Determine software object type each Matlab routine should become (library routine, kroutine, etc.)
- Convert one routine at a time to C-language
- Add Matlab C-extension (cmex) wrapper and test code in the Matlab environment
- Substitute Khoros wrapper for Matlab cmex wrapper

Integration of LLSTAP into RLSTAP, continued

Approach : calling Matlab scripts from RLSTAP

- Use Matlab Engine Library as software agent
- Determine mapping of Khoros objects to Matlab matrices
  - value segment data (e.g., cpi data cube)
  - global attributes (e.g., rlstap2_npulses
- Abstract data representation and ordering differences between Khoros and Matlab
- Write Khoros kroutines with as much general utility as feasible

Validation of RLSTAP K2

Objectives

- Check functionality of RLSTAP Khoros 2.1 version and compare with RLSTAP Khoros 1.0 version
- Focus effort on selected algorithm procedures
  - physical model baseline
  - conventional beamformer baseline
  - joint-domain optimal baseline
  - data conversion utilities (e.g., RSTERin)
- Build expertise in the integration of new routines into RLSTAP environment in support of other tasks
- Test compilation on AIX platforms

Validation of RLSTAP K2, continued

Approach

- Familiarization with RLSTAP K1 environment and baseline workspaces
- Construction of RLSTAP K2 workspaces using K1 version procedures as models
- Workspace execution and testing
- Analysis of toolbox dependencies for the incorporation of new tools into RLSTAP K2
- Generation of example kroutines utilizing RLSTAP K2 universal I/O libraries
- Compilation of RLSTAP K2 universal I/O libraries under AIX in support of Matlab Engine interface
- Full compilation under AIX in support of parallelization effort

Parallelization Tools

Motivation

- As radar sensor resolution and coverage improve, the throughput demands of radar systems increase
- STAP algorithms require extensive calculation
- Both facts drive the need for High-Performance Computing (HPC) resources for algorithm development and testing

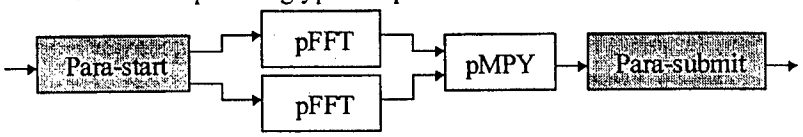
Objectives

- Provide access to massively parallel HPC resources from within the RLSTAP graphical programming environment
- Provide an interim solution that is usable immediately
- Target the IBM-SP as the primary operating environment
- Make software tools configurable for multiple platforms

Parallelization tools, continued

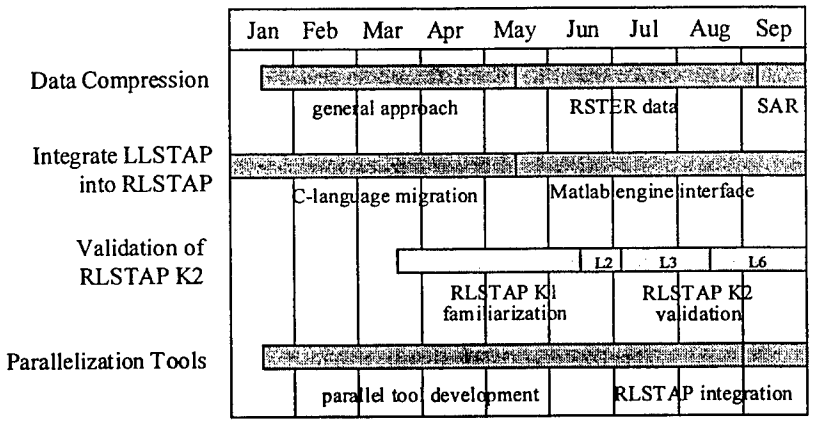
Approach

- Design control glyphs (kroutines) to delineate the beginning and end of parallel glyph compositions



- Parallel glyphs issue commands that cause remote execution of corresponding parallel algorithm functions
- Route control information between glyphs in Cantata workspace and data between agents on the parallel system
- Utilize Khoros data file format between parallel executable routines through Khoros Distributed Data Services
- Use software agents to handle redistribution of data

Task timelines



4. Critical Design Review For Advanced Signal Processing, Integration of New STAP Routines into RLSTAP

Marc Friedman
Maui High Performance Computing Center (MHPCC)
Joe Fogler
Steven Jackett
Albuquerque High Performance Computing Center (AHPCC)

7 May, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

1 Introduction

The Advanced Signal Processing (ASP) program is a DARPA sponsored activity for studying advanced processing techniques and technologies for next generation air early warning (AEW) platforms.[1] A key technology area for this activity is software tools and methodologies for collaborative algorithm development.

Khoral Research Incorporated (KRI), a spinoff company from the University of New Mexico Electrical and Computer Engineering Department, has created a software integration and development environment for information processing, data exploration and visualization called Khoros.[2] Khoros is a comprehensive software system with a rich set of tools usable by both end-users and application developers. Included in these tools is a graphical programming application called Cantata which gives users the ability to construct complex algorithms by interconnecting iconic representations, called glyphs, of processing functions in a terminal window called a workspace, using mouse point-and-click operations. Khoros has become a de-facto standard for collaborative algorithm development in the Department of Defense automatic target recognition (ATR) community.

The Rome Laboratory Space-Time Adaptive Processing (RLSTAP) tool, utilizing Khoros and its graphical programming environment Cantata, represents a state-of-the-art development environment for clutter modeling and radar simulation for advanced early warning (AEW) applications, and has found use by researchers working on Navy E-2C and the Air Force E-3A upgrades. Written initially in Khoros version 1.0, RLSTAP is currenly being ported to the latest Khoros release, version 2.1, in a separate development.

A number of space-time adaptive processing (STAP) algorithms, developed at Lincoln Laboratories, have been prototyped in the Matlab language. These algorithms are to be ported from the Matlab language to Khoros 2.1 for integration with the RLSTAP environment.

Since Matlab is primarily an interpretive language, skilled programmers seek to avoid explicit looping constructs which are generally slow. For this and other reasons, programming style differs between Matlab codes and those of compiled languages such as Khoros. This makes automated code conversion difficult and results in less than optimal memory usage and execution speed. The Lincoln Matlab scripts also utilize sparse matrix operations. These are not supported by the current generation of Matlab script compilers. Thus, the port of Matlab STAP algorithms to C-language for Khoros under this task will be performed by hand.

2 Scope

2.1 System Overview

This task will involve the integration of two types of STAP algorithms within the Lincoln Lab STAP algorithm suite – (1) Ajacent Beam Pre-Doppler STAP, and (2) Adjacent Beam Post-Doppler STAP. These two algorithms represent capabilities that do not currently exist in the RLSTAP environment. Each STAP algorithm requires approximately 20 separate functions not counting standard math library functions such as *log, exp, sqrt,* etc. Of the 20 separate functions, half are common to both algorithms.

Two forms of integration will be performed. First, an interim capability will be developed by making the current Matlab scripts for the two STAP algorithms callable from *kroutines* in Khoros 2.1 using the Matlab engine. This capability will not require any significant code conversion of the Matlab scripts, however, some data format conversion will be required for the files which must be exchanged between the Matlab and Khoros environments. These file conversions will be performed by existing kroutines in RLSTAP, specifically *RSTERin* and *RSTERout.*

[1] G. W. Titi, An Overview of the ARPA/NAVY Mountaintop Program, Proceedings IEEE Adaptive Antenna Systems Symposium, (1994).

[2] See KRI's website at *http://www.khoral.com/* for more details.

The second form of integration is the actual conversion of the two Lincoln algorithms from Matlab script to C-language code with Khoros 2.1 wrappers. Each algorithm consists of a number of routines with different levels of complexity. Some routines will become self-contained Khoros executables, whereas others may become library routines that are callable by other executables. Still others may become collections of Khoros executables. Thus each STAP algorithm routine must be categorized to determine what form of Khoros software object it should assume.

2.1.1 Khoros Routine Categories

There are four types of software routines that will be created under this development, *kroutines*, *lkroutines*, *lroutines*, and *workspaces*.

The vast majority of programs written for or supplied with the Khoros environment are *kroutines*. These include all data processing programs such as image processing and signal processing routines. Kroutines are fully usable from within the Cantata graphical programming environment, but generally do not display any graphics or images; they simply input data, process it, and output results. Kroutines may also be invoked without Cantata as ordinary applications under the Unix operating system, if desired. By convention, kroutines usually have names that begin with the letter *k* when invoked from a Unix command line, although their names may appear slightly different when represented graphically (in an iconic form called a glyph) in a Cantata workspace.

Individual *kroutines* act as drivers for library routines called *lkroutines* where I/O and calculations are actually performed. A *kroutine* usually scans its command line arguments into local variables, opens any needed input and output files, and passes file descriptors and variables to its associated library *lkroutine*. Upon return from the library routine, the files are closed in the *kroutine* before exiting.

Other library routines can be added to new or existing libraries that are not directly associated with any one *kroutine*. Such *lroutines* may be called from any number of *kroutines* within a toolbox, or even from other toolboxes.

Kroutines and other Khoros executables, represented graphically in Cantata workspaces as glyphs, can be interconnected to implement larger algorithm functionalities. These *workspaces* can be loaded and executed from Cantata using simple menu commands. An important feature of Cantata workspaces is that command line arguments (i.e. parameter settings) for the various routines are retained when workspaces are saved.

2.1.2 Algorithm Routine Categories

The STAP algorithm Matlab routines to be ported to Khoros 2.1, can be grouped into categories based on the degree to which they call other Matlab routines, the type of functionality they represent, and the amount of computation they perform.

Algorithm routines that call no other routines except intrinsic Matlab functions, which we shall call low-level routines, will be translated into Khoros lroutines.

Algorithm routines that call the lowest-level routines, but represent a small amount of computation, not warranting the development of a Khoros kroutine (and its attendant I/O overhead), will also be ported to Khoros lroutines.

Higher-level algorithm routines that may call other algorithm routines, and represent significant amounts of computation will be translated into Khoros kroutines.

The highest-level algorithm routines that perform complete algorithm functionalities will be constructed from groups of lower-level Khoros kroutines and translated into Cantata workspaces.

Some routines performs graphical operations and are strongly dependent on the Matlab environment for graphics and display. These functionalities will be performed by built-in visualization tools found in the ***Envision*** toolbox supplied with Khoros.

2.2 Limitations

The resources and time available for converting the two Lincoln STAP algorithms in their entirety by hand may not be sufficient to complete the task. Therefore, this task represents a best effort to complete the conversion of as many routines as possible using available resources within the allotted time.

The order in which the STAP algorithms will be translated will ensure that one complete algorithm functionality is obtained, as opposed to achieving partial functionality on both algorithms.

A small number of Matlab scripts included with the STAP algorithm suite perform graphics operations that are specific to the Matlab environment. Portions of the script code are implementable as built-in visualization tools that come with the Khoros distribution. Other scripts represent mixtures of calculation and graphics. The functionality of these scripts will be partitioned into separate software objects in Khoros.

The current Matlab 4.2 release, and all previous releases, perform all calculations in double precision. In the port to Khoros 2.1, calculations that can be performed as integers will be performed as integers, and all calculations that require floating-point will be performed in one type of float precision.

Since RLSTAP is currently supported under the two operating environments, AIX on IBM RS6000 computers, and SunOS or Solaris on SUN SPARC computers, the Khoros STAP routines will be designed for compatibility with these platforms. The routines may also be compatible with other platforms, but this will not be tested.

3 Reference Documents

Software routines to be written for use in Khoros will be developed using case tools built into Khoros. These include *craftsman* which is used for the creation and management of collections of routines called toolboxes, *composer* which is used to edit, manipulate, and compile existing software objects (e.g. kroutines), and *guise* which is used to edit graphical user interface (GUI) objects (e.g. panes and forms). These case tools are described in Chapters 2, 3, and 4, respectively, in the *Khoros 2.1 Toolbox Programming Guide.*

All file I/O performed by routines written in Khoros will be implemented using Khoros data services routines. These are described in the *Khoros 2.1 Data Services Guide.*

Documentation of the STAP algorithm tools will be provided in three forms – man pages, on-line help, and a printed manual.

Man pages will serve as on-line documentation for both users and programmers, describing functionality and usage of the various programs and utilities. Khoros man pages are accessable to the user via the *kman* command which is similar in operation to the standard Unix *man* command.

On-line help is accessible by the user via a *Help* button on the graphical user interface pane that can be displayed from within the Cantata graphical programming environment. The information provided through the *Help* button is similar to that obtainable from man pages.

The printed manual will provide a hardcopy representation of the documentation and encapsulate much of the documentation into an integrated whole. The tools to support printed manual documentation are embedded within the Khoros *imake* system, which provides several macros for including man pages, code segments, and function descriptions.

A description of documentation facilities within Khoros is provided in Chapter 6 of the *Khoros 2.1 Toolbox Programming Manual.* Examples of documentation generated using these built-in facilities can be found throughout that manual.

This software task will utilize existing kroutines that perform file conversion between Matlab file format files and Khoros KDF files. These routines will include *RSTERin* and *RSTERout* which are described in the on-line *man* pages of the RLSTAP_K2 environment.

4 Design

4.1 Software Development Plan

Software development will be performed in two stages. In the first stage, *kroutines* will be developed that will allow calls to the Adacent-Beam Post-Doppler *Nscript_postbU.m* and Adjacent-Beam Pre-Doppler *Nscript_prebU.m* Matlab scripts. These two algorithms will be executed as Matlab scripts using a language C coded interface which will run only the necessary computational engine components of Matlab, thus saving the overhead involved in display and human interaction.

Input and output files to these new kroutines will be in Matlab format. The existing RLSTAP kroutines *RSTERin* and *RSTERout* will be utilized to convert these files to/from Khoros format (KDF) for compatibility with other RLSTAP routines. Thus, when the new kroutines are utilized in a RLSTAP Cantata workspace, they will be bracketed by calls to *RSTERin* and *RSTERout*.

In the second stage, *workspaces*, *kroutines*, and *lkroutines* will be developed for performing STAP processing using compiled versions of the Adjacent-Beam Post-Doppler and Adjacent-Beam Pre-Doppler algorithms, converted from Matlab to the C language with Khoros 2.1 wrappers.

The development of these workspaces and routines will involve the following steps:

1. analysis and patitioning of algorithm functionalities into Khoros software object types (e.g. *kroutines*, *lkroutines*),
2. coding of C-language algorithm functions callable from Matlab,
3. testing C-language algorithm functions in Matlab,
4. migration of C-language algorithm functions to Khoros
5. coding and verification of Khoros-style test suite for ensuring correct compilation on future software installations, and
6. generation of man pages, on-line help, and printed manual documentation.

4.2 Data Description

Two distinct file types will be utilized in input/output pairs in the development. Kroutines that call the Matlab engine to execute Matlab script versions of the STAP algorithms will use a Matlab matrix file format. Input files will contain raw Coherent Pulse Interval (*CPI*) matrices and output output files will contain processed signals (*p1p_signal*). Both will contain ancillary arrays supplied to and/or generated by the algorithms. For convenience, in subsequent sections, Matlab input files are referred to as *CPI-Matrix* files, and Matlab output files that contain processed data are referred to as *SIG-Matrix* files.

Kroutines that call C-language versions of the algorithms will utilize the Khoros native *KDF* file format. On input, the *CPI* matrices will be stored in KDF value segment, and ancillary arrays will be stored in user-defined KDF attributes. On output, processed signals will be stored in the value segment, and ancillary arrays will again be stored in user-defined attributes. For convenience, in subsequent sections, KDF input files are referred to as *KDF-CPI* files and KDF output files that contain processed data are referred to as *KDF-SIG* files.

4.2.1 CPI-Matrix Files

Each CPI-matrix file is composed of one or more *cpi* matrices, and a number of ancillary arrays and scalar values, all stored as a sequence of Matlab version 4.2 matrices.

A Matlab version 4.2 matrix file contains one or more matrices each consisting of a 20-byte header, followed by a matrix name string, followed by actual matrix data. Each matrix is stored sequentially and contiguously in the file.

The Matlab header consists of five 4-byte signed integers that define, in order, (1) the matrix type, (2) number of rows in the matrix, (3) number of columns in the matrix, (4) whether the matrix is real or complex, and (5) the length of the matrix name including a NULL terminator character. The matrix type encodes the precision and data type of the matrix data, the machine architecture upon which the data was generated, and whether the matrix is sparse, numeric, or contains text. All the matrices in this application are either numeric or textual.

All matrices are stored as type double (real or complex) data values in memory within Matlab. However, to reduce storage requirements when large matrices are stored to files using the Matlab *save* command, data is converted to a data type that requires fewer bytes-per-item where possible according to an internal algorithm. For example, if all data within a real type double matrix are integral values (i.e. representable as integers), and are bounded by the representable range of 32-bit signed integers, the data is converted to and stored as 32-bit signed integers automatically when saved. If all data are integral and bounded by the representable range of 16-bit signed integers, the data is converted to and stored as 16-bit signed integers. Complex data, are similarly converted where the real and imaginary components are tested independently against the bounds.

The *cpi* arrays, which constitute a high percentage of the storage requirements of a CPI file, are complex data with real and imaginary components each stored as 32-bit signed integers. The Matlab format places all the real components of a matrix first in the file followed by all its imaginary components.

Although the number and type of ancillary arrays may vary from file to file, each file contains at least one *cpi#* array where the first array has the name *cpi1*. A typical *CPI-Matrix* file containing 5 cpi's is shown in the following table.

Matrix name	Shape	Type	Integral Values
muxtype	scalar	real	yes
trecord	scalar	real	yes
wrecord	scalar	real	yes
cpitype	vector	real	yes
cpiidx	vector	real	yes
scanidx	vector	real	yes
npulses	vector	real	yes
fxmit	vector	real	no
pri	vector	real	no
tpulse	vector	real	no
elxmit	vector	real	no
azxmit	vector	real	no
cpitime	matrix	real	no
cpi1	matrix	cplx	yes
cpi2	matrix	cplx	yes
cpi3	matrix	cplx	yes
cpi4	matrix	cplx	yes
cpi5	matrix	cplx	yes
flgRound	scalar	real	yes
flgChopTrans	scalar	real	yes
calflag	scalar	real	yes
flgEqualize	scalar	real	yes
flgCalibrate	scalar	real	yes
nCPI	scalar	real	yes
CurrFreq	scalar	real	no
ant_tilt	scalar	real	no
rster_alt	scalar	real	no
rster_lat	scalar	real	no
rster_lon	scalar	real	no
rster_pow	scalar	real	no
info	matrix	text	n/a

The number of cpi arrays in the file is stored in the scalar variable *nCPI*. The vectors *cpitype, cpiidx, scanidx, npulses, fxmit, pri, tpulse, elxmit, and azxmit* are all of length *nCPI*. The matrix *cpitime* is composed of *nCPI* rows and six columns and contains time information stored as real numeric values. The *info* array is textual and is composed of a number of rows of 80-column text strings with each character represented as an ASCII code stored one per each matrix item.

4.2.2 SIG-Matrix Files

Each SIG-matrix file is composed of a *p1p_signal* matrix, and a number of ancillary arrays and scalar values, all stored as a sequence of Matlab version 4.2 matrices.

The *p1p_signal* array, which constitutes a high percentage of the storage requirements of a SIG file, has complex data with real and imaginary components stored in type double floating-point format. The Matlab file format places all the real components of the signal in the file before all of its its imaginary components.

A typical *SIG-Matrix* file is shown in the following table.

Matrix name	Shape	Type	Integral Values
p1p_signal	matrix	cplx	no
p1p_order	vector	text	n/a
dpri_length	scalar	real	yes
del_length	scalar	real	yes
drngr_length	scalar	real	yes
dopp_values	vector	real	no
angles	vector	real	no
drngr_values	vector	real	no
nrm_typ	vector	text	n/a

The *p1p_signal* array is stored as a two-dimensional complex matrix, although it is actually a three-dimensional data cube. The *p1p_signal* data cube may be stored in one of six orientations, as specified by the order string *p1p_order*.

The *p1p_order* string can assume six values. These are assigned indices indicated in the table for convenience. The index zero is added to handle cases where an invalid order string is encountered.

Order Index	Order String
0	(invalid order string)
1	"channel sample pri"
2	"channel pri sample"
3	"sample channel pri"
4	"sample pri channel"
5	"pri sample channel"
6	"pri channel sample"

The number of rows and columns in the *p1p_signal* matrix (as stored in the Matlab file) are a function of the values *del_length*, *dpri_length*, *drngr_length* and the order string *p1p_order*, as indicated in the following table.

Order String	Row Dimension	Col Dimension
"channel sample pri"	del_length*drngr_length	dpri_length
"channel pri sample"	del_length*dpri_length	drngr_length
"sample channel pri"	drngr_length*del_length	dpri_length
"sample pri channel"	drngr_length*dpri_length	del_length
"pri sample channel"	dpri_length*drngr_length	del_length
"pri channel sample"	dpri_length*del_length	drngr_length

The length of vectors in the SIG file are indicated in the following table:

Vector	Length
p1p_order	18
dopp_values	dpri_length
angles	del_length
drngr_values	drngr_length
nrm_typ	3

The *p1p_order* and *nrm_typ* arrays are textual and are composed of ASCII codes stored as one character per each array item, with no terminating NULL character.

4.2.3 CPI-KDF Files

There are a number of differences between the Matlab matrix file format and the Khoros native KDF file format. Matlab stores data in column-major order wheras Khoros stores data in row-major order. This means that the rows and columns of two-dimensional objects are transposed. The current version of Matlab only supports one and two-dimensional arrays whereas Khoros supports arrays up to five dimensions. The Khoros value segment, where large arrays are usually stored, requires that multiple arrays be of the same dimensions, whereas in Matlab, multiple arrays are treated as separate objects and need not have the same dimensions. Matlab files can contain complex integers whereas Khoros only supports float complex and double comples data types. Moreover, Matlab stores all the real values of a complex matrix followed by its imaginary values whereas Khoros stores real and imaginary components of complex data as interleaved pairs. These differences impose a number of contraints on how the CPI data can be stored in Khoros KDF format.

Although it is technically feasible to have CPI arrays of different dimensions within the same data file, no files of this type are known to exist. Thus, CPI arrays of different sizes within the same file are considered an error condition by the RLSTAP_K2 file I/O infrastructure. This limitation is consistent with the Khoros 2.1 polymorphic model which requires that the arrays be of the same dimensions when stored in the value segment.

Although the *cpi* arrays are stored as two-dimensional matrices in Matlab, they are actually three-dimensional data cubes. Using the Khoros data services nomenclature for the dimensions of objects in the value segment, the *width, height,* and *depth* dimensions of the *cpi* arrays are computed as follows, using a combination of their Matlab dimensions and information from ancillary variables:

$$depth_i = ncols_i \tag{1}$$

$$height_i = npulses_i \tag{2}$$

$$width_i = mrows_i \div height_i \tag{3}$$

Thus, the cpi arrays are stored as a sequence of equal-sized three-dimensional data cubes in the value segment with the sequence index represented as the fourth value segment dimension called *time* in the Khoros nomenclature.

Khoros does not directly support complex integer data types. Thus, when files are converted from Matlab to Khoros KDF, complex integers must be converted to either complex float ***kcomplex*** or complex double ***kdcomplex***.

A typical set of KDF file attributes needed to store cpi files (excluding standard attributes defined for the value segment) are given in the following table. Each attribute has an implied prefix **rlstap2_** which is automatically added to the names listed in the table by RLSTAP_K2 I/O routines when operations are performed on attributes.

Variable	Attribute	Shape	Datatype
muxtype	"muxtype"	scalar	integer
trecord	"trecord"	scalar	integer
wrecord	"wrecord"	scalar	integer
cpitype	"cpitype"	vector	integer
cpiidx	"cpiidx"	vector	integer
scanidx	"scanidx"	vector	integer
npulses	"npulses"	vector	integer
fxmit	"fxmit"	vector	double
pri	"pri"	vector	double
tpulse	"tpulse"	vector	double
elxmit	"elxmit"	vector	double
azxmit	"azxmit"	vector	double
cpitime	"cpitime"	matrix	real
-	"cpitime_size"	scalar x 2	integer
flgRound	"flgRound"	scalar	integer
flgChopTrans	"flgChopTrans"	scalar	integer
calflag	"calflag"	scalar	integer
flgEqualize	"flgEqualize"	scalar	integer
flgCalibrate	"flgCalibrate"	scalar	integer
nCPI	"nCPI"	scalar	integer
CurrFreq	"CurrFreq"	scalar	double
ant_tilt	"ant_tilt"	scalar	double
rster_alt	"rster_alt"	scalar	double
rster_lat	"rster_lat"	scalar	double
rster_lon	"rster_lon"	scalar	double
rster_pow	"rster_pow"	scalar	double
info	"info"	matrix	char
-	"info_size"	scalar x 2	integer

More or fewer attributes than those listed in the table may appear in any given CPI file. Thus, the standard approach for performing I/O on attributes is to first perform an opaque copy of the input file attributes to the output file, and then modify attributes that need to be updated in the output file. This ensures that any newly-defined attributes are propagated from file to file.

4.2.4 SIG-KDF Files

The *p1p_signal* matrix will be stored in the value segment, with its real and imaginary components interleaved, as a three-dimensional object. All other matrices, representing ancillary information, will be stored as user-defined attributes.

The dimensions of the *p1p_signal* data cube, as stored in the value segment, will depend on the values of *dpri_length*, *del_length*, *drngr_length*, and *p1p_order*, according to the following table:

Order String	Width	Height	Depth
"channel sample pri"	del_length	drngr_length	dpri_length
"channel pri sample"	del_length	dpri_length	drngr_length
"sample channel pri"	drngr_length	del_length	dpri_length
"sample pri channel"	drngr_length	dpri_length	del_length
"pri sample channel"	dpri_length	drngr_length	del_length
"pri channel sample"	dpri_length	del_length	drngr_length

A typical set of KDF file attributes needed to store SIG files (excluding standard attributes

defined for the value segment) are given in the following table. As in CPI files, all attributes have the prefix rlstap2_ which is automatically added by the RLSTAP_K2 file I/O routines.

Variable	Attribute	Shape	Datatype
muxtype	"muxtype"	scalar	integer
p1p_order	"p1p_order"	vector	char
dpri_length	"dpri_length"	scalar	integer
del_length	"del_length"	scalar	integer
drngr_length	"drngr_length"	scalar	integer
dopp_values	"dopp_values"	vector	double
angles	"angles"	vector	double
drngr_values	"drngr_values"	vector	double
nrm_typ	"nrm_typ"	vector	char

The variables *p1p_order* and *nrm_typ* are stored in user-defined Khoros attributes as NULL terminated strings.

4.3 Modules

Software modules that execute the STAP algorithms as Matlab scripts are described in the following subsection. Software modules that represent Khoros C-language versions of the algorithms are described in a later subsection.

4.3.1 List of modules that execute Matlab STAP algorithms

The kroutines for executing the Adjacent-Beam Post-Doppler and Pre-Doppler algorithms are as follows:

Khoros routine	Khoros type	Comments
mxAdjBeamPostDop	kroutine	adjacent beam post doppler algorithm
mxAdjBeamPreDop	kroutine	adjacent beam pre doppler algorithm

Kroutines listed in the previous table represent driver routines that will call library routines that perform the actual work. These library routines will invoke the Matlab engine to load and execute the Matlab scripts associated with the appropriate STAP algorithms.

Library Routine	Called By	Related Matlab Script
lmxAdjBeamPostDop	mxAdjBeamPostDop	Nscript_postbU.m
lmxAdjBeamPreDop	mxAdjBeamPreDop	Nscript_prebU.m

The Matlab script *Nscript_postbU.m* will in turn call the following Matlab scripts. Calls to these scripts will be opaque to the calling *lkroutine*.

```
NSTvands.m
Npostb_Trans.m
NatrainSl.m
Npstag_adapt.m
Nwgt_norm.m
Napply_wts.m
Npostb_Trans.m
Nmbin_Trans.m
Nwgt_norm.m
Napply_wts.m
```

```
chebwgt.m
cztlds.m
dftmtx.m
rearrange_plp.m
globify_plp.m
```

The Matlab script *Nscript_prebU.m* will call the following Matlab scripts. Calls to these scripts will be opaque to the calling *lkroutine*.

```
NSTvands.m
Npreb_Trans.m
NatrainSl.m
Npreb_adapt.m
Nwgt_norm.m
Napply_wts.m
Npreb_Trans.m
Npred_trans.m
Nwgt_norm.m
Napply_wts.m
chebwgt.m
cztlds.m
dftmtx.m
rearrange_plp.m
globify_plp.m
```

Each of the scripts listed above are provided with the Lincoln Lab Matlab STAP software distribution.

4.3.2 List of modules that process CPI data

Khoros workspaces for STAP processing of CPI data using the Adjacent-Beam Pre-Doppler and Post-Doppler algorithms will include:

Khoros routine	Khoros type	Matlab script	Script option
AdjBeamPostDop.wksp	workspace	Do_script.m	Nscript_postbU.m
AdjBeamPreDop.wksp	workspace	Do_script.m	Nscript_prebU.m

Khoros kroutines for STAP processing of CPI data via the two algorithms will include:

Khoros routine	Khoros type	Matlab script	Comments
kadjbeampostdop	kroutine	Nscript_postbU.m	adj-beam post-doppler
kadjbeampredop	kroutine	Nscript_prebU.m	adj-beam pre-doppler

The above kroutines will act as drivers that call associated library lkroutines. These library routines may call lower-level library routines including routines that perform intrinsic Matlab functions.

The following two tables list the higher-level libary routines and the lower-level library routines, respectively.

Library Routine	Called By	Related Matlab Script
lkadjbeampostdop	kadjbeampostdop	Nscript_postbU.m
lkadjbeampredop	kadjbeampredop	Nscript_prebU.m

Library Routine	Called By	Related Matlab Script
lnpstagadapt	lkadjbeampostdop	Npstag_adapt.m
lnprebadapt	lkadjbeampredop	Npreb_adapt.m
lnpostbtrans	lkadjbeampostdop	Npostb_Trans.m
lnprebtrans	lkadjbeampredop	Npreb_Trans.m
lnpredtrans	lkadjbeampredop	Npred_Trans.m
lnmbintrans	lkadjbeampostdop	Nmbin_Trans.m
lnatrainsl	lkadjbeampostdop, lkadjbeampredop	NatrainSl.m
lnstvands	lkadjbeampostdop, lkadjbeampredop	NSTvands.m
lnwgtnorm	lkadjbeampostdop, lkadjbeampredop,	Nwgt_norm.m
lnapplywts	lkadjbeampostdop, lkadjbeampredop	Napply_wts.m
lchebwgt	lkadjbeampostdop, lkadjbeampredop,	chebwgt.m
lczt1ds	lchebwgt	czt1ds.m
ldftmtx	lnstvands	dftmtx.m
lrearrange	lkadjbeampostdop, lkadjbeampredop	rearrange_p1p.m
ltranspose	lrearrange, etc.	Matlab intrinsic "'"
lsort	lmprebtrans	Matlab intrinsic sort.m
+ lkfft	lczt1ds	fft.m, ifft.m (intrinsic)

The lkroutines marked with plusses in the previous table represent library routines that are included with the Khoros software distribution.

In the following subsections, each workspace, kroutine, and library routine for STAP processing is described in detail.

4.3.3 AdjBeamPostDop.wksp

Purpose:

The purpose of the workspace *AdjBeamPostDop.wksp* is to implement a adj-beam post-doppler beamformer.

Process:

AdjBeamPostDop.wksp will contain a number of interconnected glyphs that perform an equivalent functionality to the Matlab-based *Do_script.m* script executing the *Nscript_postbU.m* option.

Interfaces:

The *AdjBeamPostDop.wksp* workspace will utilize Cantata's ability to store all glyph parameters. These parameters will be entered graphically using each kroutine's pane window from within Cantata, before the workspace is created as a file.

The user will be able to execute the workspace as-is, or may edit parameters at will before execution. Detailed descriptions of the kroutine parameters are provided in the subsections that describe the individual routines.

Variable definitions:

Variables are abstracted from the user at this level. See the document subsections that describe individual routines for more information.

Routines called from AdjBeamPostDop.wksp:

AdjBeamPostDop.wksp will execute the following glyphs:

```
RSTERin (RLSTAP_K2 file I/O intrinsic)
```

```
PC      (RLSTAP_K2 pulse compression intrinsic)
kadjbeampostdop
```

4.3.4 AdjBeamPreDop.wksp

Purpose:

The purpose of the workspace *AdjBeamPreDop.wksp* is to implement a adj-beam pre-doppler beamformer.

Process:

AdjBeamPreDop.wksp will contain a number of interconnected glyphs that perform an equivalent functionality to the Matlab-based *Do_script.m* script executing the *Nscript_prebU.m* option.

Interfaces:

The *AdjBeamPreDop.wksp* workspace will utilize Cantata's ability to store all glyph parameters. These parameters will be entered graphically using each kroutine's pane window from within Cantata, before the workspace is created as a file.

The user will be able to execute the workspace as-is, or may edit parameters at will before execution. Detailed descriptions of the kroutine parameters are provided in the subsections that describe the individual routines.

Variable definitions:

Variables are abstracted from the user at this level. See the document subsections that describe individual routines for more information.

Routines called from AdjBeamPreDop.wksp:

AdjBeamPreDop.wksp will execute the following glyphs:

```
RSTERin  (RLSTAP_K2 file I/O intrinsic)
PC       (RLSTAP_K2 pulse compression intrinsic)
kadjbeampredop
```

4.3.5 kadjbeampostdop and lkadjbeampostdop

Purpose:

The purpose of *kadjbeampostdop* is to implement an adj-beam post-doppler STAP beamformer.

Process:

Kadjbeampostdop is a driver program that opens the input and output KDF files and calls the *lkadjbeampostdop* library routine where the actual work takes place.

The *lkadjbeampostdop* library routine inputs attributes and cpi data from the input file, performs the beamforming algorithm, and writes attributes and processed cpi (sig) data to the output file.

Interfaces:

The *kadjbeampostdop* routine utilizes the command line user interface (CLUI) facilities built into Khoros and detailed in the *Khoros 2.1 Toolbox Programming Guide.*

Command line arguments specific to the *kadjbeampostdop* routine are as follows:

Argument	Type	Description
-i	infile	Input data object
-o	outfile	Output data object

Variable definitions:

The *kadjbeampostdop* kroutine utilizes the variables *src* and *dest.*

The variable *src*, of type *kobject*, is a pointer to a structure returned by the call to *kpds_open_input_object* which is used to open the input file. The variable *src* is used as a file descriptor by subsequent calls to Khoros data services routines.

The variable *dest*, of type *kobject*, is a pointer to a structure returned by the call to *kpds_open_output_object* which is used to open the output file. The variable *dest* is used as a file descriptor by subsequent calls to Khoros data services routines.

Routines called from kadjbeampostdop:

The *lkadjbeampostdop* libary routine is opaque from command line execution of the *kadjbeampostdop* kroutine. Its formal parameters are as follows:

Argument	Type	Description
src	kobject	Input object descriptor
dest	kobject	Output object descriptor

Other routines called by library routine *lkadjbeampostdop* are:

`lnpstagadapt`
`lnpostbtrans`
`lnatrainsl`
`lnstvands`
`lchebwgt`
`lnwgtnorm`
`lnapplywts`
`ldftmtx`
`lcz1ds`
`lrearrange`

These routines are described in later sections.

4.3.6 kadjbeampredop and lkadjbeampredop

Purpose:

The purpose of *kadjbeampredop* is to implement an adj-beam pre-doppler STAP beamformer.

Process:

Kadjbeampredop is a driver program that opens the input and output KDF files and calls the *lkadjbeampredop* library routine where the actual work takes place.

The *lkadjbeampredop* library routine inputs attributes and cpi data from the input file, performs the beamforming algorithm, and writes attributes and processed cpi (sig) data to the output file.

Interfaces:

The *kadjbeampredop* routine utilizes the command line user interface (CLUI) facilities built into Khoros and detailed in the *Khoros 2.1 Toolbox Programming Guide.*

Command line arguments specific to the *kadjbeampredop* routine are as follows:

Argument	Type	Description
-i	infile	Input data object
-o	outfile	Output data object

Variable definitions:

The *kadjbeampredop* kroutine utilizes the variables *src* and *dest.*

The variable *src*, of type *kobject*, is a pointer to a structure returned by the call to *kpds_open_input_object* which is used to open the input file. The variable *src* is used as a file descriptor by subsequent calls to Khoros data services routines.

The variable *dest*, of type *kobject*, is a pointer to a structure returned by the call to *kpds_open_output_object* which is used to open the output file. The variable *dest* is used as a file descriptor by subsequent calls to Khoros data services routines.

Routines called from kadjbeampredop:

The *lkadjbeampredop* libary routine is opaque from command line execution of the *kadjbeampredop* kroutine. Its formal parameters are as follows:

Argument	Type	Description
src	kobject	Input object descriptor
dest	kobject	Output object descriptor

Other routines called by library routine *lkadjbeampredop* are:

`lnprebadapt`
`lnprebtrans`
`lnpredtrans`
`lnatrainsl`
`lnstvands`
`lchebwgt`
`lnwgtnorm`
`lnapplywts`
`ldftmtx`
`lczt1ds`
`lrearrange`
`lsort`

These routines are described in later sections.

4.3.7 lnpstagadapt

Purpose:
The library routine *lnpstagadapt* implements the functionality of the Matlab script *Npstag_adapt.m.*
Process:
This routine implements PRI-staggered STAP transform domain adaptive nulling.
Interfaces and variable definitions:
Formal parameters for the *lnpstagadapt* routine are given in the following table:

Argument	Type	Description
del_length	integer	delay length
dpri_length	integer	pri length
drngr_length	integer	drngr length
p1p_signal	kdcomplex *	signal data cube
iorder	integer	input order index
T	double *	pstag transf
V	double *	S-T sv's
train	integer *	trn gate list
ndop	integer	length of dopp_values array
dload	double	power in dB
weights	double *	data cube of computed weights

Other routines called:

lkrearrange

4.3.8 lnprebadapt

Purpose:

The library routine *lnprebadapt* implements the functionality of the Matlab script *Npreb_adapt.m.*

Process:

This routine implements adj-Beamspace STAP transform domain adaptive nulling. It is almost identical in function to *lnpredadapt.*

Interfaces and variable definitions:

Formal parameters for the *lnprebadapt* routine are given in the following table:

Argument	Type	Description
del_length	integer	delay length
dpri_length	integer	pri length
drngr_length	integer	drngr length
p1p_signal	kdcomplex *	signal data cube
iorder	integer	input order index
T	double *	pstag transf
V	double *	S-T sv's
train	integer *	trn gate list
nsubc	integer	number of pulses per sub-CPI
tmp_wts	double *	temporal taper weights
p1p_prf	double	pulse reptition frequency
dload	double	power in dB
sub_tpr	double *	sub-CPI sv taper (e.g. binomial)

Other routines called:

`ldftmtx`
`lkrearrange`

4.3.9 lnpostbtrans

Purpose:

The library routine *lnpostbtrans* implements the functionality of the Matlab script *Npostb_Trans.m.*

Process:

This routine generates the characteristic STAP transformation for the adjacent-beam post-Doppler (multi-bin) algorithm.

Interfaces and variable definitions:

Formal parameters for the *lnpostbtrans* routine are given in the following table:

Argument	Type	Description
dpri_length	integer	pri length
del_length	integer	delay length
nsubP	integer	number of Doppler bins per bin subset
nsubB	integer	number of beams per beam subset
tmp_wts	double *	temporal taper weights
p1p_prf	double	pulse reptition frequency
V	double *	beam set
lk_idx	integer *	flag vector
T	double *	returns pstag transf array
dopp_values	double *	returns dopp values array

Other routines called:

`lnmbintrans`
`lsort`

4.3.10 lnprebtrans

Purpose:
The library routine *lnprebtrans* implements the functionality of the Matlab script *Npreb_Trans.m.*
Process:
This routine generates the characteristic STAP transformation for the adjacent-beam pre-Doppler algorithm.
Interfaces and variable definitions:
Formal parameters for the *lnprebtrans* routine are given in the following table:

Argument	Type	Description
dpri_length	integer	pri length
del_length	integer	delay length
nsubP	integer	number of Doppler bins per bin subset
nsubB	integer	number of beams per beam subset
V	double *	beam set
lk_idx	integer *	flag vector
T	double *	returns pstag transf array

Other routines called:

`lnpredtrans`
`lsort`

4.3.11 lnpredtrans

Purpose:
The library routine *lnpredtrans* implements the functionality of the Matlab script *Npred_Trans.m.*
Process:
This routine generates the characteristic STAP transformation for the pre-Doppler algorithm.
Interfaces and variable definitions:
Formal parameters for the *lnpredtrans* routine are given in the following table:

Argument	Type	Description
dpri_length	integer	pri length
del_length	integer	delay length
nsubc	integer	number of sub CPIs
T	double *	returns pstag transf array

Other routines called:

(none)

4.3.12 lnmbintrans

Purpose:
The library routine *lnmbintrans* implements the functionality of the Matlab script *Nmbin_Trans.m.*
Process:
This routine generates the characteristic STAP transformation for the multi-bin post-Doppler algorithm.
Interfaces and variable definitions:
Formal parameters for the *lnmbintrans* routine are given in the following table:

Argument	Type	Description
dpri_length	integer	pri length
del_length	integer	delay length
nsubc	integer	number of sub-CPIs
tmp_wts	double *	temporal taper weights
p1p_prf	double	pulse reptition frequency
T	double *	returns pstag transf array
dopp_values	double *	returns dopp values array

Other routines called:

`ldftmtx`

4.3.13 lnatrainsl

Purpose:
The library routine *lnatrainsl* implements the functionality of the Matlab script *NatrainSl.m.*
Process:
This routine constructs a sliding training set to fix existing data with supplied parameters.
Interfaces and variable definitions:
Formal parameters for the *lnatrainsl* routine are given in the following table:

Argument	Type	Description
begin_rng	double	begin range of interest (km)
end_rng	double	end range of interest (km)
nom_grd	integer	guard band gates
nom_trn	integer	trn rgn, no. samples (even)
nom_rgn	integer	tgt rgn width
drngr_values	double *	drngr value array
train	double *	training set array

Other routines called:

(none)

4.3.14 lnstvands

Purpose:
The library routine *lnstvands* implements the functionality of the Matlab script *NSTvands.m.*
Process:
This routine generates Space-Time vandermonde steering vectors for specified look angles.
Interfaces and variable definitions:
Formal parameters for the *lnstvands* routine are given in the following table:

Argument	Type	Description
tdlam	double	time delay parameter
angles	double *	array of angles
dpri_length	integer	dpri length
del_length	integer	del length
spat_wts	double *	spatial weights
V	double *	vandermonde vectors

Other routines called:

4.3.15 lnwgtnorm

Purpose:
The library routine *lnwgtnorm* implements the functionality of the Matlab script *Nwgt_norm.m.*
Process:
This routine normalizes space-time weights.
Interfaces and variable definitions:
Formal parameters for the *lnwgtnorm* routine are given in the following table:

Argument	Type	Description
weights	double *	weight array [input and output]
T	double	time period
ndop	integer	number of Doppler bins
nrm_flg	char *	normalization flag ["CGN" or "CGT"]
spat_wts	double *	spatial weights
V	double *	vandermonde vectors

Other routines called:

4.3.16 lnapplywts

Purpose:
The library routine *lnapplywts* implements the functionality of the Matlab script *Napply_wts.m.*
Process:
This routine applies space-time weights to space-time data.
Interfaces and variable definitions:
Formal parameters for the *lnapplywts* routine are given in the following table:

Argument	Type	Description
weights	double *	S-T weight array
T	double	time period
ndop	integer	number of Doppler bins
train	integer *	train table
drngr_values	double *	drngr values
p1p_signal	kdcomplex *	p1p signal matrix
order	integer *	order index
del_length	integer *	del length
drngr_length	integer *	drngr length
dpri_length	integer *	dpri length

Other routines called:

lkrearrange

4.3.17 lchebwgt

Purpose:
The library routine *lchebwgt* implements the functionality of the Matlab script *chebwgt.m.*
Process:
This routine calculates the Chebyshev weights using a chirp-z transform.
Interfaces and variable definitions:
Formal parameters for the *lchebwgt* routine are given in the following table:

Argument	Type	Description
nel	integer	desired length
slob	double	side-lobe level in dB
wgt	double	array of Chebychev weights (normalized)

Other routines called:

`cztlds`

4.3.18 lcztlds

Purpose:
The library routine *lcztlds* implements the functionality of the Matlab script *cztlds.m.*
Process:
This function calculates Chebychev weights for either odd or even numbers of weights using the chirp-z transform.
Interfaces and variable definitions:
Formal parameters for the *lcztlds* routine are given in the following table:

Argument	Type	Description
array	double *	input sequence
nx	integer	input length
nupts	integer	output length
umin	double *	begining of chirp-z transform
umax	double *	ending of chirp-z transform
beta	double	beta parameter
isign	integer	sign parameter
wgt	double *	output sequence of length nupts

Other routines called:

lkfft
lkifft

4.3.19 ldftmtx

Purpose:
The library routine *ldftmtx* implements the functionality of the Matlab script *dftmtx.m.*
Process:
This function forms a NxN Discrete Fourier Transform matrix consisting of values around the unit-circle whose inner product with a column vector of length N yields the discrete Fourier Transform of the vector.
Interfaces and variable definitions:
Formal parameters for the *lcztlds* routine are given in the following table:

Argument	Type	Description
n	integer	size of square matrix
b	double *	returns DFT matrix values

Other routines called:

(none)

4.3.20 krearrange and lkrearrange

Purpose:

The purpose of *krearrange* is to rearrange the orientation of one or more three-dimensional data cubes located in the value segment of a KDF file. This routine works for both SIG and CPI files.

Process:

Krearrange is a driver program that opens the input KDF file for reading using the the Khoros data services *kpds_open_input_object* kroutine, and opens the output KDF file object using the the Khoros data services *kpds_open_output_object* routine. The user can specify, via a command line parameter, whether the input orientation of the three-dimensional cubes is to be obtained from an examination of the "order" attribute of the input file, or from a user specified order. A second command line parameter specifies the desired output order of the three-dimensional data cubes.

The actual work is performed by the library routine *lkrearrange* which accepts the input and output file descriptors and command line options from *krearrange*, performs the operations, and returns.

First the attributes of the input object are copied to the output object, with the size information updated to reflect the new data cube orientation. Then, the data is read, processed, and written one three-dimensional cube at a time.

Interfaces:

The *krearrange* routine utilizes the command line user interface (CLUI) facilities built into Khoros. Details of these automated facilities and capabilities are provided in the *Khoros 2.1 Toolbox Programming Guide.*

Command line arguments specific to the *krearrange* routine are as follows:

Argument	Type	Description
-i	infile	Matrix input data object
-o	outfile	Resulting output data object
-oorder	list	output order index [1 ... 6]
[-iorder]	list	input order index [0, 1, ... 6]
	default	0 = derive from object

For the meaning of order indices in the range [1 ... 6] in this context, see the table of order indices in Section 4.2.1 of this document.

Variable definitions:

The *krearrange* kroutine utilizes the variables *src*, *iorder*, *oorder*, and *dest.*

The variable *src* of type *kobject* is used by Khoros data services routines to access the input file object.

The integer variable *iorder* is used to specify the input order of the data cubes in the value segment. If *iorder* is zero (default), the input order is to be obtained from the input object "order" attribute. If *iorder* is not zero, it is used as an assumed input order.

The integer variable *oorder* is used to specify the desired output order of the data cubes in the value segment.

The variable *dest* of type *kobject* is a pointer to a structure returned by the call to *kpds_open_output_object* and is used as a file descriptor by Khoros data services routines to access the output file.

Routines called from krearrange:

The *lkrearrange* library routine is opaque from command line execution of the *krearrange* kroutine. Its formal parameters are as follows:

Argument	Type	Description
src	kobject	Input object descriptor
iorder	integer	Input order index [0, 1, ... 6
oorder	integer	Ouput order index [1 ... 6]
dest	kobject	Output object descriptor

4.3.21 ltranspose

Purpose:

The purpose of library routine *ltranspose* is to perform a matrix transpose of a two-dimensional contiguous memory array in-place. The array may be of any fixed-size data type.

Process:

The *ltranspose* routine uses a permutation-by-cycle algorithm to perform an in-place matrix transpose of an array of any fixed-size data type.

Reference: *"Computational Frameworks for the Fast Fourier Transform", by Charles Van Loan, (SIAM) 1992.*

Interfaces and variable definitions:

Formal parameters for the *ltranspose* routine are given in the following table:

Argument	Type	Description
si	integer	size of each datum in bytes
n1	integer *	ptr to size of matrix in memory-fast dimension
n2	integer *	ptr to size of matrix in memory-slow dimension
x	kaddr	ptr to contiguous array of data in memory

The matrix dimensions are supplied to *ltranspose* as pointers to permit the dimensions to be updated to match the new matrix shape.

Other routines called:

ltranspose calls no other routines.

4.3.22 lsort

Purpose:

The library routine *lsort* sorts each row of a matrix of type double elements along the width (memory-fast) dimension. Each item is sorted in ascending order. An optional result matrix can be generated whose items are sorted in slave fashion to the first matrix. This feature can be used to produce an aray of sorted indices that correspond to the change in order of the first matrix.

Process:

This routine uses a heap sort algorithm. A quicksort algorithm would be slightly faster in some cases, but would require additional memory.

Interfaces and variable definitions:

Formal parameters for the *lsort* routine are given in the following table:

Argument	Type	Description
nw	integer	size of matrix in width (memory-fast) dimension
nh	integer	size of matrix in height (memory-slow) dimension
x	double *	ptr to input matrix x
y	double *	ptr to output matrix y
z	double *	ptr to optional output matrix z, or NULL

Other routines called:

Routine *lsort* calls no other routines.

5 Test Plan

Two types of tests will be performed on the kroutine software modules, validation test suites, and acceptance tests.

5.1 Validation Test Suites

Validation test suites will be created for each kroutine using the Khoros non-interactive test suite generation infrastructure, described in Chapter 7 of the *Khoros Toolbox Programming Guide.*

The purpose of these test suites is to provide a reproducible, non-interactive method of algorithm verification that helps guarantee the integrity, robustness, and portability of the code. This is particularly useful when the kroutines are recompiled on a new machine architecture.

A test shell script will be written for each kroutine based upon a test script template provided with the Khoros software distribution.

Small data files will be generated within or loaded from these shell scripts to provide input file stimuli for the tests. There will be no dependencies of these test scripts on other toolboxes except for the FFT routine *lkfft* which is required for certain calculations.

These shell scripts will be located in a *testsuites* subdirectory to be placed under the toolbox directory.

5.2 Acceptance Tests

The following tests will be performed to demonstrate the functionality of the toolbox:

The Adjacent-Beam Post-Doppler Matlab script will be called from a Khoros workspace and the results will be written to a file. The Matlab environment will be used to view the results for acceptance test purposes.

The Adjacent-Beam Pre-Doppler Matlab script will be called from a Khoros workspace and the results will be written to a file. The Matlab environment will be used to view the results for acceptance test purposes.

A workspace will be run that executes the Adjacent-Beam Post-Doppler algorithm. The resultant signal will be compared with the result generated by the Matlab version.

A similar workspace will be run that executes the Adjacent-Beam Pre-Doppler algorithm, if completed. The resultant signal will be compared with the result generated by the Matlab version.

5. RLSTAP_HPC Integrations of Algorithms Written in MATLAB

Steve Jackett
Albuquerque High Performance Computing Center (AHPCC)

30 September, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

Outline

- **Introduction**
- Porting Matlab scripts to Khoros executables
- Calling Matlab scripts directly from Khoros
- Application examples
- Summary

Objectives

- Provide access to Space-Time Adaptive Processing (STAP) algorithms written in Matlab script from the Khoros graphical programming environment *Cantata* and RLSTAP
  - Investigate porting STAP algorithm Matlab scripts to C language with Khoros wrapper, as compiled executables
  - MHPCC focus is on automated conversion tools from Matlab script to C or C++ (e.g. cross compilers)
  - AHPCC focus is on hand conversion from Matlab scripts to C for Khoros integration
  - Develop an interface for calling STAP algorithm Matlab scripts directly from Khoros without code conversion
  - STAP algorithm Matlab scripts utilized as is, with possible minor modification to top-level script interface
  - inclusion as Khoros toolbox

Matlab script code conversion: challenges

- Matlab data representation differs substantially from that of Khoros data representation
  - Matlab accesses two-dimensional arrays in column-major order whereas C language Khoros executables access such arrays in row-major order (FORTRAN vs. C data ordering)
  - Matlab stores complex data with all real values followed by all imaginary values whereas Khoros executables store complex data with real and imaginary values interleaved.
- Code optimization strategies differ significantly between typical programmers in the two domains
  - Matlab: Much effort is expended employing high-level Matlab intrinsic functions in clever ways to eliminate explicit looping constructs
  - Khoros/C: Focus is more on using appropriate data types and pointer schemes for efficient access of multi-dimensional arrays

Matlab script direct execution from Khoros: challenges

- Matlab computational engine interface limited to one or two-dimensional objects (version 4.2.c and earlier)
  - RLSTAP data objects can have more than two dimensions
- Matlab computational engine interface limited to four input and four output matrix objects (determined by experimentation)
  - RLSTAP data objects can have tens of attributes in addition to value data
  - Each attribute becomes a separate matrix in Matlab
- Matlab data representation differs from that of Khoros
  - Complex data pairs not interleaved
  - Data ordering is column major
  - All data is type double, including strings (version 4.2c)
  - Matrices have names (limited to 19 characters)
- Matlab computationl engine has minimal error reporting

Outline

- Introduction
- **Porting Matlab scripts to Khoros executables**
- Calling Matlab scripts directly from Khoros
- Application examples
- Summary

Porting Matlab scripts to C (Khoros) executables: Hand conversion

- Determine code hierarchy, dependencies, and formal parameters of each Matlab script routine
- Identify Matlab scripts that represent functions already implemented by existing Khoros toolboxes
- Determine Khoros software object type each Matlab routine should become (library routine, kroutine, xvroutine, etc).
- Convert one routine at a time to C language - lowest levels first
- Place a Matlab C extension (cmex) wrapper around converted and compiled routine
- Substitute Khoros wrapper for Matlab cmex wrapper
- Test and debug

Porting Matlab scripts to Khoros executables: Example

C language translation of Matlab script [y,w] = redopp(x,v)

```
#include <stdio.h>
...
int redopp(
  unsigned int wx,  /* 'fast' dimension of matrix x */
  unsigned int hx,  /* 'slow' dimension of matrix x */
  double *x,        /* input matrix x */
  unsignd int nv,   /* length of optional input vector v */
  double *v,        /* input vector v, or NULL */
  unsigned int r,   /* replication factor r */
  double *y,        /* output matrix y */
  double *w)        /* optional output vector w, or NULL
*/
{
  ...
}
```

Matlab function arguments mapped to formal parameters in C

Porting Matlab scripts to Khoros executables: Example

```
#include <mex.h>

#define X_IN     phrs[0]
#define V_IN     prhs[1]
#define Y_OUT    plhs[0]
#define W_OUT plhs[1]

void mexFunction(
  int nlhs,
  Matrix *plhs[],
  int nrhs,
  Matrix *prhs[])
{
  x = mxgetPr(X_IN);
  w = mxCreateFull(nv, 1, REAL);
  ...
/* C-language 'redopp' routine */
  (void)redopp(mx, nx, x, nv, v, 5, y, w);}
```

typical Matlab cmex wrapper for C routine

Porting Matlab scripts to Khoros executables: Example

Khoros wrapper for C 'redopp' routine

```
/*
 * Khoros: $Id$
 */
...
#include "internals.h"
...
int redopp(
  unsigned int wx, unsigned int hx,
  double *x,
  unsigned int nv, double *v,
  unsigned int r, double *y, double *w)
{
  ...
}
```

Khoros wrapper for low-level C routine can be as simple as adding a header

Khoros data services calls occur at higher-level software layer

Matlab algorithm scripts re-coded to C

- Matlab instrinsics (w/ no corresponding C library support)
  - sort*, interp, transpose, filter, hsv2rgb
- Utilities implemented in place of intrinsics
  - hsv_to_rgb, hsort, hsort2, isort, isort2
- Low-level STAP scripts
  - hsv, vibgyor, vibgyor2, zeropad, zeropadm
- Mid-level STAP scripts
  - redopp, rearrange_plp, get_slice, cfar_normGO

* The Matlab intrinsic sort function has more options than standard C library routine including the ability to condition the sorting of one vector on another.

Hand conversion of Matlab to C: level of effort

- Conversion of one line of Matlab script takes roughly 1-2 hours, assuming
  - programmer is proficient in Matlab and C-language
  - defensive programming strategies are utilized
  - some knowledge of signal processing algorithms
- Time required can vary tenfold depending on the experience & proficiency of the programmers involved
- Major issues
  - data ordering complexities, especially 3-D objects
  - understanding intent of 'clever' Matlab code
  - software strategies that lead to debugging pitfalls

Matlab script conversion: lessons learned

- The hand conversion process from Matlab script is
  - laborious
  - error prone
  - very time consuming
- However, once a working compilable source is available, migration to Khoros libraries and kroutines is relatively simple
- A seamless test environment is vital
  - Use of a C-extension (cmex) wrapper allows routines to be individually tested against their Matlab counterparts.
- Matlab scripts that convert to library routines require little or no modification for incorporation into Khoros since Khoros-dependent aspects can be relegated to higher software layers.

Outline

- Introduction
- Porting Matlab scripts to Khoros executables
- **Calling Matlab scripts directly from Khoros**
- Application examples
- Summary

Calling Matlab scripts directly from Khoros

- Software agent is the Matlab Engine Library (C language) which includes routines such as
  - engOpen: starts up the Matlab computational engine
  - engClose: shuts down the Matlab computational engine
  - engPutMatrix: sends a matrix to the Matlab engine from local process memory
  - engGetMatrix: gets a matrix from the Matlab engine to local process memory
- The Matlab computational engine executes as a separate process in the background without a user interface
- The Matlab Engine Library communicates with the Matlab computational engine on Unix machines using pipes.

Matlab Engine Library interface to Khoros

- The Matlab Engine Library is simple to use with Khoros software objects once the proper toolbox dependencies have been established
  - Additions similar to the following must be made to the 'toolbox.def' file in the user's toolbox repository
    - TOOLBOX_INCLUDE +<= -I/usr/local/matlab/extern/include
    - TOOLBOX_LIBDIR +<= -L/usr/local/matlab/extern/lib/ibm_rs
    - SYS_LIBRARIES +<= -libm_rs
- The Matlab Engine Library can then be compiled and linked in the usual Khoros manner

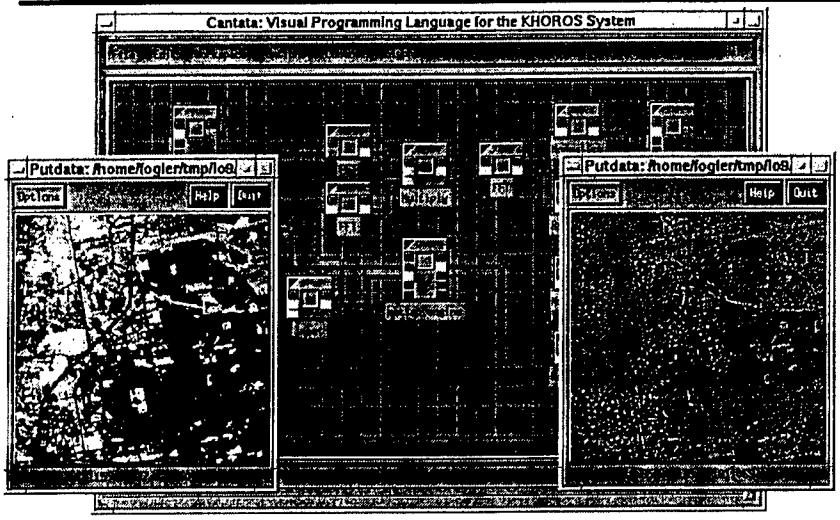
Khoros software tools for executing Matlab scripts

- Generic **Matlab_exec** glyph that inputs up to four data objects, sends them to the Matlab engine, executes one or more Matlab scripts and returns up to four data objects
  - One matrix is extracted from each input object (value segment)
  - One matrix result is written to each output object
  - No attributes are propagated from input to output beyond those associated with the value segment
- Specialized **Matlab_stap** glyph that invokes STAP algorithms
  - Inputs single Khoros data object
  - Sends data cube in value segment as matrix
  - Sends all rlstap2 attributes as separate matrices
  - Returns data cube and updated attributes to Khoros object
- Pane objects that invoke **Matlab_stap** glyph for specific algorithms
  - **mxPreDopStap** invokes pre-doppler STAP script
  - **mxPostDopStap** invokes post-doppler STAP script

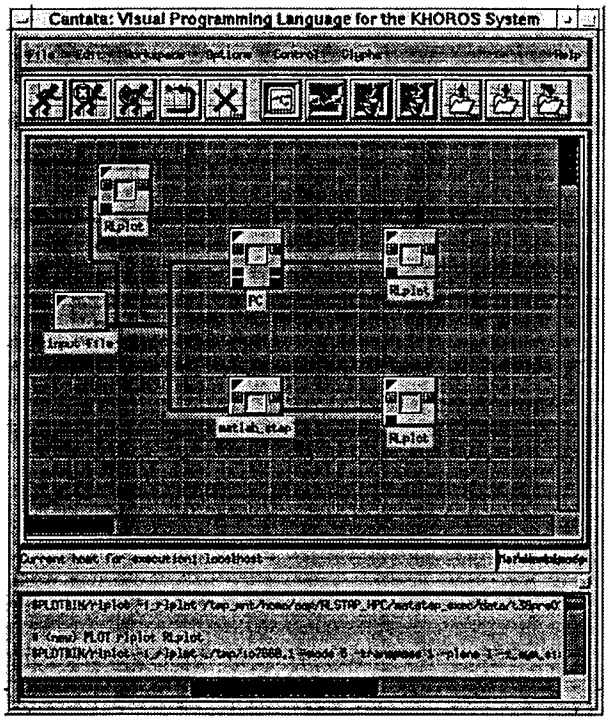
Outline

- Introduction
- Porting Matlab scripts to Khoros executables
- Calling Matlab scripts directly from Khoros
- **Application examples**
- Summary

Matlab_exec application: Frequency domain convolution



Matlab_stap application: RLSTAP pulse compress example



- In this example, the **Matlab_stap** glyph invokes a Matlab pulse compression script
- Pulse compression is also invoked using a native RLSTAP glyph
- The two results are passed to display glyphs for comparison

Summary

- Hand conversion of code from Matlab to Khoros executable is a difficult, error prone, and time consuming process
- Conversion can be performed at a rate of one Matlab script line every 1 or 2 hours, at best
- Testing of individual converted routines is facilitated by using Matlab C extension wrappers
- The Matlab Engine facilities can be utilized to interface Matlab and Khoros programs in fairly general ways
- High-level software tools have been written and demonstrated that invoke algorithm functions written in Matlab script from Khoros, including STAP related algorithms

6. Critical Design Review For Advanced Signal Processing, Integration of Parallelism, and RLSTAP

Marc Friedman
Maui High Performance Computing Center
Joe Fogler
Paul Alsing
Ruth Klundt
Ken Summers
Albuquerque High Performance Computing Center (AHPCC)

30 April, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

1 Introduction

The Advanced Signal Processing (ASP) program is a DARPA sponsored activity for studying advanced processing techniques and technologies for next generation air early warning (AEW) platforms.[1] A key technology area for this activity is software tools and methodologies for collaborative algorithm development.

Khoral Research Incorporated (KRI), a spinoff company from the University of New Mexico Electrical and Computer Engineering Department, has created a software integration and development environment for information processing, data exploration and visualization called Khoros.[2] Khoros is a comprehensive software system with a rich set of tools usable by both end-users and application developers. Included in these tools is a graphical programming application called Cantata which gives users the ability to construct complex algorithms by interconnecting iconic representations, called glyphs, of processing functions in a terminal window called a workspace, using mouse point-and-click operations. Khoros has become a de-facto standard for collaborative algorithm development in the Department of Defense automatic target recognition (ATR) community.

The Rome Laboratory Space-Time Adaptive Processing (RLSTAP) tool, utilizing Khoros and its graphical programming environment Cantata, represents a state-of-the-art development environment for clutter modeling and radar simulation for advanced early warning (AEW) applications, and has found use by researchers working on Navy E-2C and the Air Force E-3A upgrades. Written initially in Khoros version 1.0, RLSTAP is currenly being ported to the latest Khoros release, version 2.1, in a separate development.

Signal processing algorithms required by AEW applications are computationally intensive and utilize large data sets for experimentation and validation. These are driving a need for distributed parallel processing resources such as those available at the Maui High Performance Computing Center (MHPCC).

Khoral Research Inc. has begun the development of an advanced version of Khoros that will directly support parallel algorithm development and experimentation within the Cantata graphical programming environment. This development will involve substantial modification to the infrastructure of Khoros and will take a few years to complete.

The purpose of the effort described in this document is to provide an interim capability for parallel algorithm development utilizing a new set of software tools that will work within the existing Khoros 2.1 environment. These tools will provide a standard and easy-to-use mechanism for incorporating parallel algorithms into the RLSTAP environment.

2 Scope

2.1 System Overview

Software tools will be developed under this task for the purpose of facilitating the incorporation of parallel algorithms in the RLSTAP environment under Khoros 2.1. The target computing environment will be the IBM Parallel Operating Environment (POE) on the IBM SP (RS/6000) architecture running under the AIX operating system. Interprocess communication will be performed utilizing the Message Passing Interface (MPI).

[1] G. W. Titi, An Overview of the ARPA/NAVY Mountaintop Program, Proceedings IEEE Adaptive Antenna Systems Symposium, (1994).

[2] See KRI's website at *http://www.khoral.com/* for more details.

2.1.1 Motet

The software tools will be based on a paradigm for integrating MPI-based parallel library routines into the Khoros environment called *Motet.* [3] However, some extensions will be made to the *Motet* architecture to increase its usability in the RLSTAP environment. The enhanced version of *Motet* to be implemented under this task, will be designated *RLSTAP_HPC* .

Motet acts as a batch parallel job processor, utilizing a collection of interconnected Khoros glyphs, called a Motet composition, to construct a shell script containing command lines for parallel job submission and execution. A Motet composition is always bracketed by two special glyphs called *Motet_Start* and *Motet_Submit. Motet_Start* establishes the beginning of a Motet composition and *Motet_Submit* delineates the end of a Motet composition. Glyphs that lie between the *Motet_Start* and *Motet_Submit* represent the parallel calculations to be performed. An example Cantata workspace containing a Motet composition with three calculation kRoutine is shown in Figure 1. This workspace also contains glyphs for converting data between Khoros Data Format (KDF) and Motet file format. These conversion functions are typically performed by the the *Motet_Start* and *Motet_Submit* glyphs, but are shown explicitly in the figure for clarity.

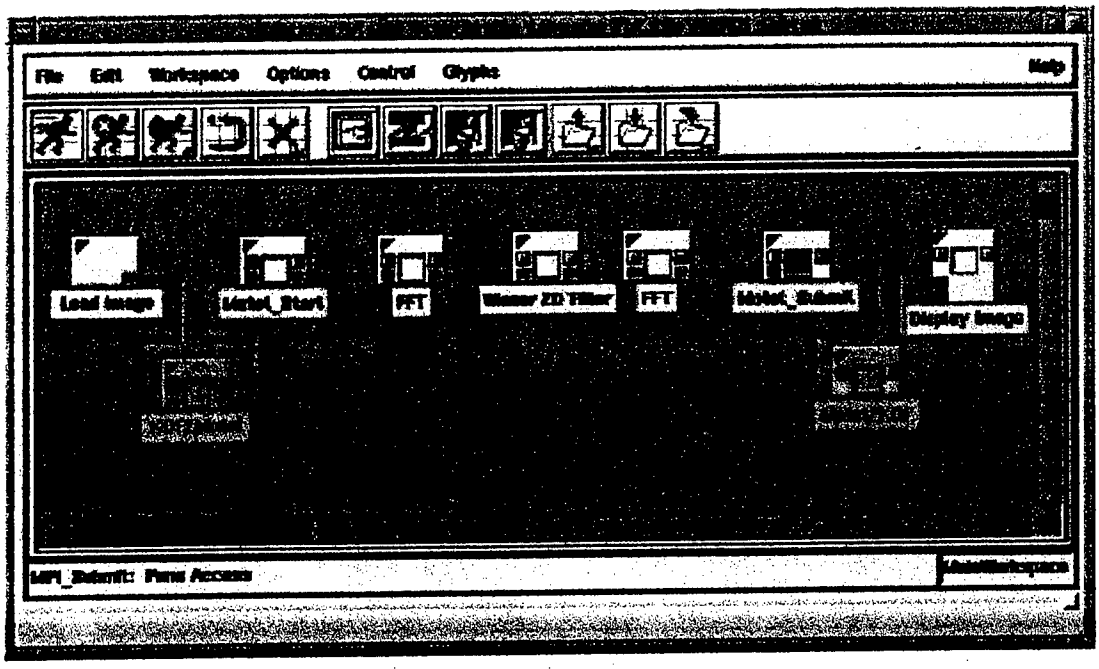


Figure 1. Example workspace containing a Motet composition.

There are three primary components to Motet *kRoutines*, *pkRoutines*, and *lpkRoutines*.

kRoutines execute serially on the local machine from within Cantata. These include the mandatory control kRoutines *Motet_Start* and *Motet_Submit*, as well as various calculation kRoutines that cause the remote execution of parallel algorithm functions.

pkRoutines execute remotely in response to instructions generated by calculation kRoutines in the Motet composition. *pkRoutines* perform parallel data distribution operations for data preparation and call library routines that perform the actual algorithm computations. *pkRoutines* utilize Motet facilities but have no Khoros code.

[3] *Motet* was developed by Karen G. Haines at the University of New Mexico Albuquerque Resource Center under the direction of Dr. Thomas P. Caudell.

lpkRoutines are MPI-based library routines that perform algorithm functions. They contain no Motet or Khoros code.

The sequence of events that occur in the execution of a Motet composition in the IBM POE Load Leveler environment is as follows. First Cantata executes the *Motet_Start* glyph on the local machine. Then *Motet_Start* reads its command line arguments (i.e. the state of its pane window), converts its input file from KDF format to Motet format, and initiates the construction of a load leveler shell script. *Motet_Start* then exits and Cantata executes the next kRoutine in the chain, a calculation kRoutine.

The calculation kRoutine adds a line to the load-leveler shell script for executing an algorithm function. In addition, the calculation kRoutine may perform a one-time KDF to Motet file conversion on any secondary inputs that it might obtain from serial glyphs in the Cantata workspace. Cantata then executes the remaining calculation kRoutines in the composition in proper sequence, with each kRoutine adding a line to the load-leveler script for execution of an algorithm function.

Finally, Cantata executes the *Motet_Submit* kRoutine which adds a line to the load-leveler shell script to gather results from the nodes, and closes the script. *Motet_Submit* then submits the shell script to load-leveler for execution and waits for completion. As the shell script executes, each parallel executable *pkRoutine* in the script is invoked which in turn reads its command line arguments to obtain filenames and parameters, performs any necessary data distributions, calls its associated *lpkroutine* to perform an algorithm calculation, and writes the results to an output file before exiting.

Once the shell script has fully executed and the job completes, *Motet_Submit* converts any result data from Motet format to KDF format on the local machine and passes the data to the next glyph in the local Cantata workspace where serial execution resumes.

2.1.2 RLSTAP_HPC

The RLSTAP_HPC system will follow the basic structure of Motet with the following changes and additions.

In *RLSTAP_HPC* , the Khoros KDF file format will be used directly, eliminating the need for file format conversions. The Khoros parallel data services library will be employed for file I/O and data distribution.

RLSTAP_HPC will also provide a level of abstraction of the parallel operating environment from the calculation kRoutines. These kRoutines will be implemented as Khoros *pane* objects that call a common *launch* script. The *launch* script will abstract the operating environment from the kRoutines.

The *Par_Start* and *Par_Submit* (which correspond to Motet_Start and Motet_Submit in Motet) kRoutines will be implemented as *kroutine* objects. In order to support multiple input and output data files, *Par_Start* and *Par_Submit* will be modified in the RLSTAP_HPC implementation, to support multiple input and output glyph connections with matching underlying communication fabric to the parallel world. By routing all connections between serially executing glyphs and RLSTAP_HPC kRoutines via the *Par_Start* and *Par_Submit* glyphs, synchronization of time-varying data exchanged between the serial and parallel worlds can be ensured.

A workspace utilizing the proposed extensions to Motet (RLSTAP_HPC) is shown in Figure 2. In the example, *Par_Start* and *Par_Submit* are capable of establishing up to four simultaneous connections between the surrounding serially executing glyphs and the RLSTAP_HPC composition, although only two input and two output connections are utilized.

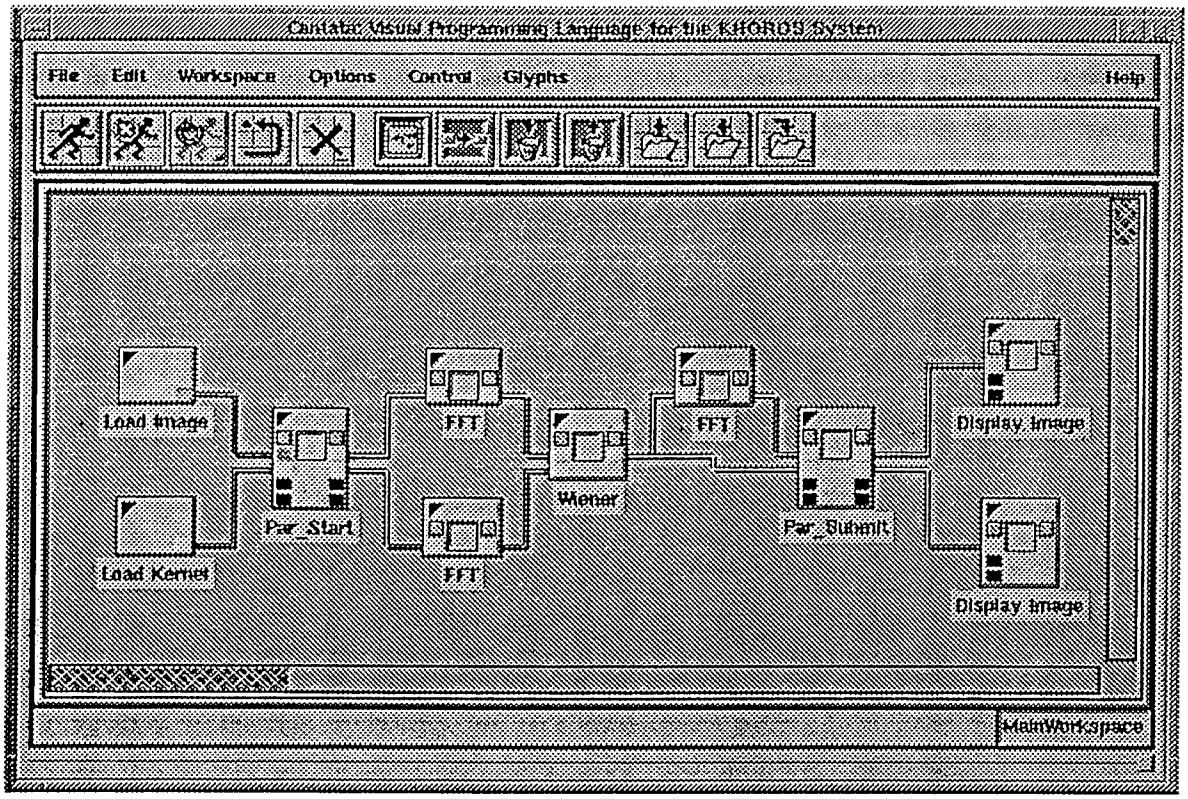


Figure 2. Example workspace using proposed extensions to Motet (RLSTAP_HPC).

In the RLSTAP_HPC architecture, the user will have the option of executing the composition multiple times from the same parallel job. This will be accomplished by constructing a loop in the load-leveler shell script that repeatedly executes the parallel routines in sequence each time new input data becomes available from the preceeding serial glyphs. The *Par_Start* and *Par_Submit* glyphs will utilize a continuous run mechanism available under Khoros 2.1 when operated in this mode.

2.1.3 Data Distribution

RLSTAP_HPC will provide facilities for the distribution of data between parallel executables. Data distribution is performed via a library call at the beginning of each *pkRoutine*. The user simply specifies the desired data distribution required by the associated *lpkRoutine* algorithm function, and upon return from the library call, receives a KDF object containing the distributed portion of the data for the current node. The *lpkRoutine* can then be called to perform the algorithm function on the data in memory.

After the algorithm function is performed, a final library call is made to write the data to an output file. By default, the data distribution type of the output file is specified to be that which resulted from the algorithm function.

Algorithm functions are commonly implemented using scientific libraries such as *PESSL* and *Scalapack*. These libraries may perform MPI-based data redistributions in the course of algorithm calculations (within the same pkRoutine) of which RLSTAP_HPC will be unaware. RLSTAP_HPC need only be informed of the distribution type of the final results in order to write them correctly to KDF files before the *pkRoutine* exits.

Algorithm functions could conceivably utilize the RLSTAP_HPC data distribution operations for redistributing data in the course of algorithm calculations within the same pkroutine. However, RLSTAP_HPC data distribution adds an extra (unnecessary) layer when called from inside the MPI program. Data distribution of memory-based data could therefore be performed with greater efficiency using strictly MPI-based operations such as those found in the scientific libraries.

RLSTAP_HPC will support the data distribution types shown in the following table:

Dimensionality	Distribution Type	Comments
1-D	distribute(*)	broadcast 1-D
1-D	distribute(block)	vector into sub vectors
2-D	distribute(*,*)	broadcast 2-D
2-D	distribute(block,*)	distribute along first dimension
2-D	distribute(*,block)	distribute along second dimension
3-D	distribute(*,*,*)	broadcast 3-D
3-D	distribute(block,*,*)	distribute along first dimension
3-D	distribute(*,block,*)	distribute along second dimension
3-D	distribute(*,*,block)	distribute along third dimension
4-D	distribute(*,*,*,*)	broadcast 4-D
4-D	distribute(block,*,*,*)	distribute along first dimension
4-D	distribute(*,block,*,*)	distribute along second dimension
4-D	distribute(*,*,block,*)	distribute along third dimension
4-D	distribute(*,*,*,block)	distribute along fourth dimension
5-D	distribute(*,*,*,*,*)	broadcast 5-D
5-D	distribute(block,*,*,*,*)	distribute along first dimension
5-D	distribute(*,block,*,*,*)	distribute along second dimension
5-D	distribute(*,*,block,*,*)	distribute along third dimension
5-D	distribute(*,*,*,block,*)	distribute along fourth dimension
5-D	distribute(*,*,*,*,block)	distribute along fifth dimension

2.2 Limitations

The primary purpose of this task is to provide a set of software tools to facilitate the incorporation of parallel versions of RLSTAP algorithms into Khoros. In addition, example workspaces and routines will be provided that demonstrate the use of these software tools on a limited set of algorithm functions. However, these functions will not represent complete RLSTAP algorithms.

A two-dimensional parallel FFT algorithm implementation and three STAP-related algorithm functions will be addressed – *InvCovar*, *STAPwgts*, and *Covar*. Due to tight schedule contraints, only the FFT algorithm and one of the other three functionalities may be implementable within the alotted time. However, a best effort will be made to implement them all.

RLSTAP_HPC represents an interim solution for incorporating parallel algorithms into the RLSTAP environment. It will provide an easy-to-use mechanism that will function within Khoros 2.1 without any modifications to the Khoros infrastructure. However, it cannot be expected to have the level of functionality achievable by adding distributed parallel computing facilities directly into the Khoros infrastructure. Thus, it is expected that users will eventually migrate their RLSTAP_HPC compositions to KRI's own implementation of parallel distributed computing in Khoros when it becomes available.

RLSTAP_HPC will be implemented to work in the IBM Parallel Operating Environment (POE) using Load Leveler on the IBM SP (RS/6000) architecture running under the AIX operating system. Although *RLSTAP_HPC* will contain abstractions to facilitate the porting of parallel codes to other architectures, the actual port to other environments will not be performed.

RLSTAP_HPC will be capable of gathering and scattering KDF format files from the Cantata workspace. Although Khoros parallel data services is capable of distributing different types of

segments within KDF files (e.g. *geometry*, *mask*, and *value*), only data distribution utilizing the *value* segment and *user-defined attributes* will be demonstrated under this task. Use of the *value* segment will impose the requirement for distributed data of a single data type of regular shape in up to five dimensions in any given file. The *user-defined attributes* may be of any type and number, but will be distributed globally and gathered from the root node as defined within the capabilities of Khoros parallel data services.

RLSTAP_HPC will support a variety of block-type distributions on data in up to five dimensions. However, not every type of block distribution possible on five dimensional data will be implemented. The distributions to be supported in the development were selected based on the anticipated needs of STAP algorithms in the ASP project.

3 Reference Documents

The proposed *RLSTAP_HPC* architecture is based on the original *Motet* paradigm for integrating MPI parallel library routines into Khoros. An overview of the *Motet* architecture can be found in the document *Motet: A Paradigm for Integrating MPI Parallel Library Routines Into Khoros, K. G. Haines and P. Alsing, University of New Mexico, Albuquerque, New Mexico.*

Software routines to be written for use in Khoros will be developed using case tools built into Khoros. These include *craftsman* which is used for the creation and management of collections of routines called toolboxes, *composer* which is used to edit, manipulate, and compile existing software objects (e.g. kroutines), and *guise* which is used to edit graphical user interface (GUI) objects (e.g. panes and forms). These case tools are described in Chapters 2, 3, and 4, respectively, in the *Khoros 2.1 Toolbox Programming Guide.*

Data distribution will be implemented using a pre-release of Khoros paraserv (parallel data services). See Khoros documentation on paraserv for more information.

Documentation of the data compression tools will be provided in three forms – man pages, on-line help, and a printed manual.

Man pages will serve as on-line documentation for both users and programmers, describing functionality and usage of the various programs and utilities. Khoros man pages are accessable to the user via the *kman* command which is similar in operation to the standard Unix *man* command.

On-line help is accessible by the user via a *Help* button on the graphical user interface pane that can be displayed from within the Cantata graphical programming environment. The information provided through the *Help* button is similar to that obtainable from man pages.

The printed manual will provide a hardcopy representation of the documentation and encapsulate much of the documentation into an integrated whole. The tools to support printed manual documentation are embedded within the Khoros *imake* system, which provides several macros for including man pages, code segments, and function descriptions.

A description of documentation facilities within Khoros is provided in Chapter 6 of the *Khoros 2.1 Toolbox Programming Manual.* Examples of documentation generated using these built-in facilities can be found throughout that manual.

4 Design

4.1 Software Development Plan

Software development will involve the following steps. They are largely chronological in order, although there may be some overlap, particularly in the development of the data distribution routines.

1. Create installation instructions for KRI parallel distribution services *message* and *paraserv* toolboxes along with a simple example using the RLSTAP_HPC paradigm.

2. Add parallel operating environment abstractions to Par_Start and Par_Submit by introducing calls to separate command scripts.

3. Add support for up to four input and output connections between Par_Start and Par_Submit and surrounding serial glyphs in support of single-pass execution of static input data.

4. Develop calculation kRoutine template(s) containing parallel operating environment abstractions and support for multiple input and output connections.

5. Construct an example workspace that includes a parallel FFT routine which utilizes a parallel scientific library such as PESSL.

6. Develop pkroutine template(s) for a subset of input and output connection combinations.

7. Construct a Joint Domain Optimum workspace containing parallelized Inverse Covariance, STAP weights, and Covariance kRoutines.

8. Add inter-process communication (IPC) features to Par_Start, Par_Submit, and their associated command scripts to synchronize multiple input and output connections.

9. Add continuous run features to Par_Start, Par_Submit, and their associated command scripts to support multi-pass execution on single batch jobs.

10. Generate man pages, on-line help, and printed manual documentation.

4.2 Data Description

Data will be in standard Khoros KDF format. After data distribution the distributed KDF objects will also contain a new polymorphic attribute which controls the data distribution. It is defined as follows:

`KPDS_DISTRIBUTION`

Dimension This argument describes which dimension of the polymorphic value segment will be distributed over the different processors. Only one dimension can be distributed. The distribution along this dimension will occur as dictated by the type argument.

Type This argument describes how the data will be distributed over the different processors.

Communication group This argument specifies the communication group over which the data is distributed.

4.3 Modules

The software modules in RLSTAP_HPC consist of the control kRoutines *Par_Start* and *Par_Submit*, calculation kRoutine templates, and pkroutine templates. and routines for automated data distribution.

4.3.1 Par_Start

Purpose:

The purpose of Par_Start is to initialize an RLSTAP_HPC series of parallel computations. All Cantata input connections from serially executed glyphs to a parallel lineup are routed to the inputs of Par_Start.

Process:

Par_Start reads its command line arguments, and creates the initialization and definition portion of a Load-Leveler shell script file in accordance with those arguments. It also creates a local database file and places information about its input and output connections in that file. This information is utilized by subsequent glyphs in the parallel lineup to generate filenames for proper connectivity among their pkRoutine parallel executable counterparts.

Interfaces:

The *Par_Start* routine utilizes the command line user interface (CLUI) facilities built into Khoros. These facilities provide automatic code generation of usage reporting, and support the input of command line via a graphical interface in Cantata graphical programming environment. Details of these automated facilities and capabilities are provided in the *Khoros 2.1 Toolbox Programming Guide.*

Command line arguments specific to the *Par_Start* routine are as follows:

Argument	Type	Description
-inData	infile	Input file 1 from previous glyph
-inData2	infile	Input file 2 from previous glyph
-inData3	infile	Input file 3 from previous glyph
-inData4	infile	Input file 4 from previous glyph
-outData	outfile	Output file 1
-outData2	outfile	Output file 2
-outData3	outfile	Output file 3
-outData4	outfile	Output file 4
-initialDir	string	initial directory (e.g. $HOME)
-jobName	string	parallel job name (e.g. pk_job1)
-nNodes	integer	number of Parallel nodes (e.g. 4)
-execMode	list	1 = "single-pass", 2 = "continuous"
-class	list	1 = "short", 2 = "mixed", 3 = "long", 4 = "bigmem", 5 = "large", 6 = "medium", 7 = "small_long", 8 = "small_short"
-notifyUser	string	user e-mail address: $USER
-notification	list	1 = "always", 2 = "complete", 3 = "error", 4 = "never", 5 = "start"

Variable definitions:

The *Par_Start* routine utilizes the variables *inDataFileName, inDataFileName2, inDataFileName3,* and *inDataFileName4* to store the input filenames for input data connections, and the variables *outDataFileName, outDataFileName2, outDataFileName3,* and *outDataFileName4* to store the output filenames for output data connections.

Two other files are required for operation of *Par_Start.* A database file is used to retain data connection filenames and parallel operating environment information required by subsequent calculation glyphs as well as *Par_Submit.* The Khoros *kdbm* database routines are used to create and access this file. The filename for this database file is constructed from the user-specified *jobName* as *[jobName].dbm,* and is stored in the variable *dbmFileName.* A struct of type *kdbm* and given the name *dbm* is returned from a call to the Khoros *kdbm_open* routine and used to reference the database file in subsequent database calls from within *Par_Submit.* Two struct variables are used to store information in the database file, *key* and *val* of type *kdatum.*

Additional information stored in the database includes the number of processor nodes *nNodes,* and a scalar called *currentProcessNum* which is initialized to the value 1 and is incremented by each calculation kRoutine in the parallel lineup . The *currentProcessNum* value is checked by *Par_Submit* to ensure that at least one calculation kRoutine exists in the composition. Another parameter stored in the database is *execMode* which is used to inform *Par_Submit* whether the parallel job is to execute continuously in a loop, or just once. If continous operation is requested, Par_Start creates an *until-do* loop in the shell script. This loop is completed by Par_Submit by placing a *done* line at the end of the same script.

The second file required for operation of *Par_Start* is a shell script used to construct the sequence of commands for parallel execution by the parallel lineup . This file is created by *Par_Start* and

given the filename *[jobName].sh*, which is stored in the variable *shellFileName*.

Since the *jobName* is used in the construction of the database and shell script filenames, it must be made available to subsequent kRoutines in the parallel lineup who need access to these files. Thus, *jobName* and its associated directory name, are written to each of the output connections (e.g. outData, outData2) of the Par_Start glyph.

4.3.2 Par_Submit

Purpose:

The purpose of Par_Submit is to initiate remote parallel execution of a parallel lineup . It also gathers results from the parallel computations and makes them available to glyphs that follow in the workspace.

Process:

Par_Submit accesses the Load-Leveler shell script file and database file created by Par_Start by constructing their filenames using the jobName and directory path information received via its Cantata input connection *inData*.

Par_Submit completes the Load-Leveler shell script that was created by Par_Start and appended by intervening calculation kRoutines, and submits the script for parallel execution. Par_Submit then waits for the parallel job to complete. If the *execMode*, stored in the database by Par_Start, is set for single-pass operation, Par_Submit executes the script, waits for the job to complete, gathers the results, and then exits. If the *execMode* is set for continous opereration, Par_Submit appends a *done* line to complete the shell script loop initiated by Par_Start, submits the script for execution, and then repeatedly gathers results from the parallel calculations and makes them available to following glyphs in the workspace until the parallel job is terminated by Par_Start. This mode of operation is similar to Cantata's continuous run feature, for local execution of serial glyphs.

Interfaces:

The *Par_Submit* routine utilizes the command line user interface (CLUI) facilities built into Khoros. These facilities provide automatic code generation of usage reporting, and support the input of command line via a graphical interface in Cantata graphical programming environment. Details of these automated facilities and capabilities are provided in the *Khoros 2.1 Toolbox Programming Guide*.

Command line arguments specific to the *Par_Submit* routine are as follows:

Argument	Type	Description
-inData	infile	Input file 1 from previous glyph
-inData2	infile	Input file 2 from previous glyph
-inData3	infile	Input file 3 from previous glyph
-inData4	infile	Input file 4 from previous glyph
-outData	outfile	Output file 1
-outData2	outfile	Output file 2
-outData3	outfile	Output file 3
-outData4	outfile	Output file 4

Variable definitions:

The *Par_Submit* routine utilizes the variables *inDataFileName, inDataFileName2, inDataFileName3,* and *inDataFileName4* to store the input filenames for input data connections, and the variables *outDataFileName, outDataFileName2, outDataFileName3,* and *outDataFileName4* to store the output filenames for output data connections.

Two other files are required for operation of *Par_Submit*. A database file is used to retain data connection filenames and parallel operating environment information required by preceeding calculation glyphs as well as *Par_Start*. The Khoros *kdbm* database routines are used to access this file. The filename for this database file is constructed from the user-specified *jobName* as

[jobName].dbm, and is stored in the variable *dbmFileName*. A struct of type *kdbm* and given the name *dbm* is returned from a call to the Khoros *kdbm_open* routine and used to reference the database file in subsequent database calls from within *Par_Submit*. Two struct variables are used to store information in the database file, *key* and *val* of type *kdatum*.

Additional information available from the database includes the number of processor nodes *nNodes*, and a scalar called *currentProcessNum* which is initialized to the value 1 by Par_Start, and is incremented by each calculation kRoutine in the parallel lineup . If the *currentProcessNum* value is equal to 1 when examined by Par_Submit, an error is reported indicating that there were no calculation kRoutines present in the Motet composition.

The second file required for operation of *Par_Submit* is a shell script used to form the sequence of commands for parallel execution by the parallel lineup . This file is created by *Par_Start* and given the filename *[jobName].sh*, which is stored in the variable *shellFileName*.

4.3.3 kRoutine Templates

Purpose:

The purpose of kRoutine Templates is to provide the user examples of how to construct calculation kRoutines for their applications. Three kRoutine Templates will be created for the following I/O topologies:

Template	I/O Connections
kRoutine11	One input and one output
kRoutine21	Two inputs and one output
kRoutine22	Two inputs and two outputs

These templates will be in the form of Khoros kroutines and/or pane objects that can be copied from the RLSTAP_HPC toolbox to the user's toolbox using the Khoros case tool *Craftsman*. The user need only add the the name of their parallel executable *pkroutine* and additional command line options via the Khoros case tool *Guise*.

Process:

A kRoutine Template (or any other kRoutine) obtains the *jobName* from its first input connection, and constructs filenames for the database and shell script created by Par_Start. It then accesses the database to obtain the output filenames of preceeding kRoutines that it will use as inputs in the parallel environment. It also constructs its own output filenames of the form *jobName.kRoutine.out* and places them in the database for use by subsequent kRoutines in the parallel lineup . Finally, it appends a parallel executable command line to the shell script that includes its parallel executable name (e.g. pkfft), input and output filenames, and command line options.

Interfaces:

The kRoutine Templates utilizes the command line user interface (CLUI) facilities built into Khoros and detailed in the *Khoros 2.1 Toolbox Programming Guide*.

Command line arguments for the kRoutine22 template are as follows:

Argument	Type	Description
-i1	infile	Input file 1
-i2	infile	Input file 2
-o1	outfile	Output file 1
-o2	outfile	Output file 2
-r	string	Parallel executable name [default: pNoop]

The command line arguments for kRoutine11 and kRoutine21 are similar except for the number of input and output file arguments.

Variable definitions:

The kRoutine Template routines utilize the variables *inDataFile1*, *inDataFile2*, and so on, to store the input file names. The variables *outDataFile1*, *outDatafile2*, and so on, are used to store the output file names.

The parallel executable name associated with the kRoutine is stored in the variable *pExecName*.

Additional variables may be added by the user to store command line options specific to the algorithm function of the kRoutine.

Routines called from kRoutine Templates

The parallel executable name specified by the *-r* command line option is caused to execute when the shell script is eventually submitted by Par_Submit, although it is not directly called by the kRoutine.

In RLSTAP_HPC , the possibility of making kRoutines Khoros *pane* objects that call a single *launch* routine will be investigated. However, some additional experimentation will be required to determine whether this is feasible. The impact of making kRoutines pane objects will be to reduce the memory requirments of kRoutines by eliminating the need for multiple kroutines.

4.3.4 pkRoutine Templates

Purpose:

The purpose of pkRoutine Templates is to provide the user examples of how to construct parallel executable driver routines for their applications. Three pkRoutine Templates will be created for the following I/O topologies:

Template	I/O Connections
pkRoutine11	One input and one output
pkRoutine21	Two inputs and one output
pkroutine22	Two inputs and two outputs

These templates will be in the form of pkroutines that can be copied from the RLSTAP_HPC toolbox to the user's toolbox. The user need only define the type of desired data distribution required on input to their associated *lpKroutine* at the top of the template, and insert the name of their executable *lpKroutine* in the middle of the template.

Process:

pKroutines are driver programs that utilize library calls to perform data distribution and read on input, call associated *lpKroutines* that perform the actual algorithm computations on data in local memory, and then write the calculation results from memory to output files again using the library.

The pKroutines can also obtain algorithm parameters and options via command line arguments which are scanned and passed along with the data buffers and data size information to the lpKroutines.

Interfaces:

The pKroutine Templates utilize a simple command line user interface to obtain input and output filenames as well as algorithm options and parameters required by their associated lpKroutines.

Command line arguments used by the pkRoutine22 template are as follows:

Argument	Type	Description
-i1	infile	Input file 1
-i2	infile	Input file 2
-o1	outfile	Output file 1
-o2	outfile	Output file 2

The input and output files utilized by the pkRoutines are read and written from disks local to the individual processor nodes. Thus, each pkRoutine executed on each node accesses its own data using the same filenames as its conterparts on other nodes.

Variable definitions:

The pkRoutine Template routines utilize the variables *inDataFile1*, *inDataFile2*, and so on, to store the input file names. The variables *outDataFile1*, *outDatafile2*, and so on, are used to store the output file names.

Additional variables may be added by the user to store command line options specific to the algorithm function of the lpKroutine.

Routines called from pkRoutine Templates

pkRoutine Templates (and pkRoutines in general) call algorithm library routines called *lpkRoutines*. These library routines perform the actual algorithm functions on data in local memory. They typically utilize parallel scientific libraries such as PESSL.

Library routines utilized by pkRoutines to distribute, read and write data, are described in the following subsections.

4.3.5 lPar_distribute

Purpose:

The purpose of *lPar_distribute* is to obtain data from an input file with a user-defined data distribution. If the input file is not already distributed, or has a different data distribution than that requested by the user, the data is re-distributed accordingly before it is presented to the caller.

Process:

This routine utilizes a number of lower-level library routines to read the data file headers and data, and to perform any needed redistributions. One call to *lPar_distribute* must be made for each input data file to the calling *lkRoutine*.

Interfaces and variable definitions:

Formal parameters for the *lPar_distribute* routine are given in the following table:

Argument	Type	Description
ndim	integer	number of dimensions in data
distType	integer	data distribution type
fname	char *	filename of object to be distribued
dataPtr	void *	pointer to distributed data

4.3.6 lPar_writeData

Purpose:

The purpose of *lPar_writeData* is to write intermediate output data to a file on the nodes (usually at the end of a pkroutine). This file will be the input data for the next pkroutine in the parallel lineup.

Process:

This routine utilizes lower level library routines to write the distribute data to a file. The same name will be used for each distributed portion of the data on each node.

Interfaces and variable definitions:

Formal parameters for the *lPar_writeData* routine are given in the following table:

Argument	Type	Description
fname	char *	filename of object to be written
dataPtr	void *	pointer to distributed data

4.3.7 lPar_gatherFile

Purpose:

The routine *lPar_gatherFile* gathers a distributed file and writes it to a single file.

Process:

The routine *lPar_gatherFile* checks the distribution type of the distributed data and calls the appropriate lower level library routines to gather the data. The gathered data is written to a file.

Interfaces and variable definitions:

Formal parameters for the *lPar_gatherFile* routine are given in the following table:

Argument	Type	Description
dataPtr	void *	distributed data
rank	integer	processor rank
nNodes	integer	size of comm group
filename	char *	output KDF filE

5 Test Plan

Two types of tests will be performed on the kroutine software modules, validation test suites and acceptance tests.

5.1 Validation Test Suites

Validation test suites are a set of test routines that follow the design methodology of the Khoros 2.1 test suite facilites (see Chapter 7 of the *Khoros Toolbox Programming Guide*). These are deliverables and will be provided with any routine written for Khoros. Their purpose, as outlined in the Khoros documentation, is to help ensure that when a khoros toolbox is compiled on a new architecture, it is functioning properly. Khoros–type test suites follow a certain style. First, they are non-interactive. This permits test suites to be run on entire Khoros installations with results catalogued to log file(s). Second, they utilize small amounts of data, often generated on-the-fly. This keeps the installation from bloating to unrealistic proportions, and also allows each routine to execute in a small amount of time. Thus, Khoros style test suites are not meant to do exhaustive testing, just simple verification. A test shell script will be written for each kroutine in RLSTAP_HPC .

5.2 Acceptance Tests

The RLSTAP_HPC toolbox will be demonstrated by implementing three specific algorithm functions from RLSTAP in parallel and demonstrating their operation in a Khoros workspace. The results of the parallel implementation of the algorithm functions will be compared to the results of the serial implementation.

Acceptance tests are a one-time event in which it is shown by example that a pre-chosen algorithm piece selected within RLSTAP has been successfully parallelized and it produces correct results. The deliverable part of the acceptance test is a Cantata Workspace that runs the acceptance test example.

In this case a workspace that includes a parallelized FFT, and a second workspace that includes parallelized versions of one or more of the algorithm functionalities *Covar*, *STAPWgts*, and *InvCovar*, will be delivered for acceptance tests.

7. RLSTAP_HPC Parallelization Effort

Joe Fogler
Ruth Klundt
Ken Summers
Albuquerque High Performance Computing Center (AHPCC)

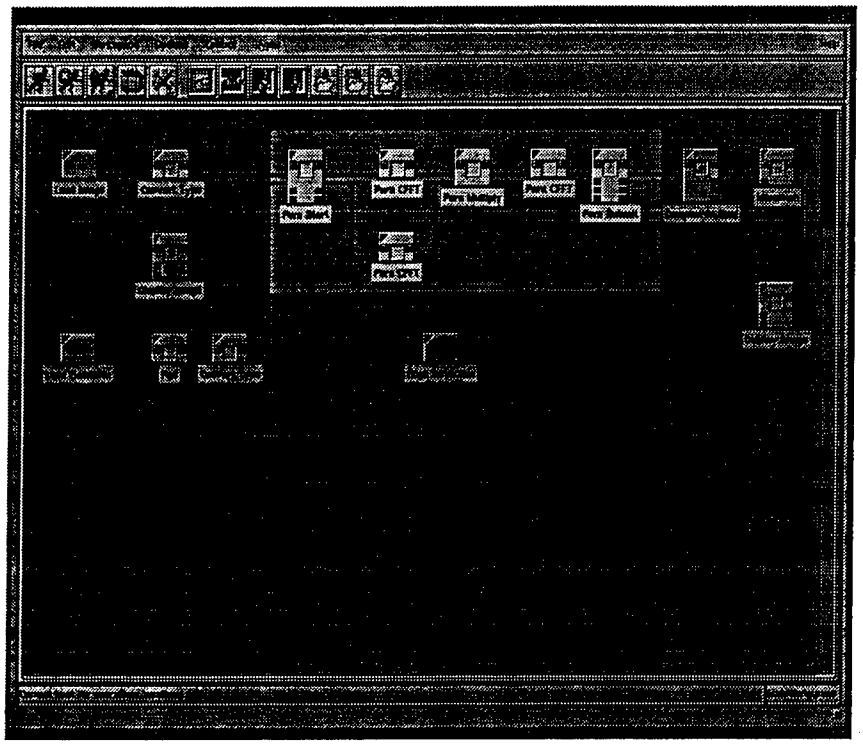
30 September, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

Objectives

- Provide access to massively parallel computing systems from within the Khoros graphical programming environment *Cantata*
- Allow the creation of simple-to-use parallel glyphs
- Provide an interim solution that is usable immediately
- Anticipate and simplify software migration to future versions of Khoros that will support parallelism
- Target the IBM SP at MHPCC as primary operating environment
- Make software tools configurable for multiple platforms

The solution: RLSTAP_HPC



Para Start: designates the beginning of a parallel lineup
Parallel Glyphs: one for each executable
Para Submit: designates the end of a parallel lineup

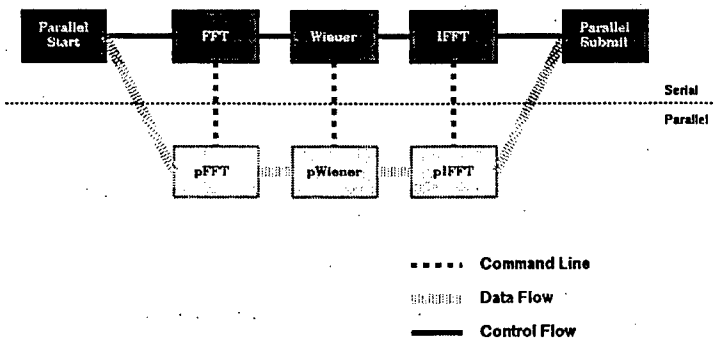
Outline

- Introduction
  - Objectives
  - System Overview
- **System Design**
  - Features
  - Parallel program flow
  - Software layers
  - Platform independence
- Applications
  - Parallel image processing
  - Parallel RLSTAP routines

RLSTAP_HPC system design features

- Platform executing Cantata is independent of parallel platform
- Layered software structure
- Software abstractions achieve parallel platform independence
- Uses Khoros data file format between parallel executable routines through Khoros Distributed Data Services
- Parallel glyph behavior follows Khoros paradigm

Parallel program data flow



- *Control* flows between glyphs in Cantata workspace
- *Data* flows between agents on the parallel system
- Agents handle the redistribution of data

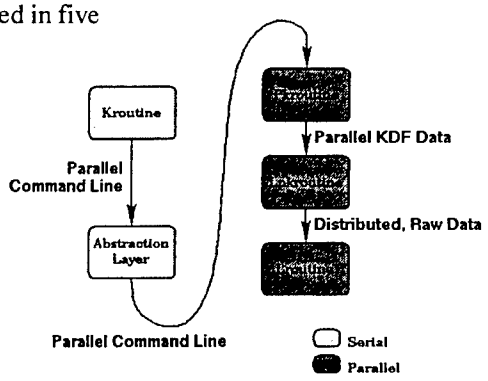
Parallel program execution flow

- Para_Start
  - Initializes the parallel environment
- Parallel algorithm glyphs
  - Each parallel glyph generates a command line for execution of parallel agents
- Para_Submit
  - Gathers distributed data from last parallel glyph output
  - Finalizes parallel environment

Software layers

RLSTAP_HPC is implemented in five software layers

- Serial layers
  - Kroutine
  - Abstraction layer
- Parallel layers
  - Pkroutine
  - Lpkroutine
  - Lroutine



Kroutine layer (serial platform)

- Constructed using Khoros standard CASE tools (i.e. *Craftsman* and *Composer*)
- Reads control information from the previous parallel glyph or Parallel Start
- Generates parallel commands to perform an associated parallel function on the remote parallel platform
- Passes on control information to the next parallel glyph

Abstraction layer (serial platform)

- Interface between kroutine and parallel layers
- Encapsulates platform-specific information necessary to run parallel jobs
- Takes commands generated by kroutine and uses them to create parallel command lines
  - in batch mode, commands may be collected for submission
  - in interactive mode, commands may be immediately launched

Pkroutine layer (parallel platform)

- The pkroutine is the top-level of the parallel agent associated with the serial kroutine
- Performs functions that are analogous to the kroutine on the serial platform
- Parses the command line received from the abstraction layer, opens the data object, and passes it to the next layer
- This layer can be migrated to track future developments in Khoros top-level kroutine design with minimal impact on other layers

Lpkroutine layer (parallel platform)

- Checks the distribution state of the data object received from pkroutine
- Distributes or re-distributes the data if necessary
- Reads data into memory and passes data pointer and parameters to *lroutine*

Lroutine layer (parallel platform)

- The *lroutine* performs the actual parallel computation
- Works on data stored in memory
- Unaware of Khoros

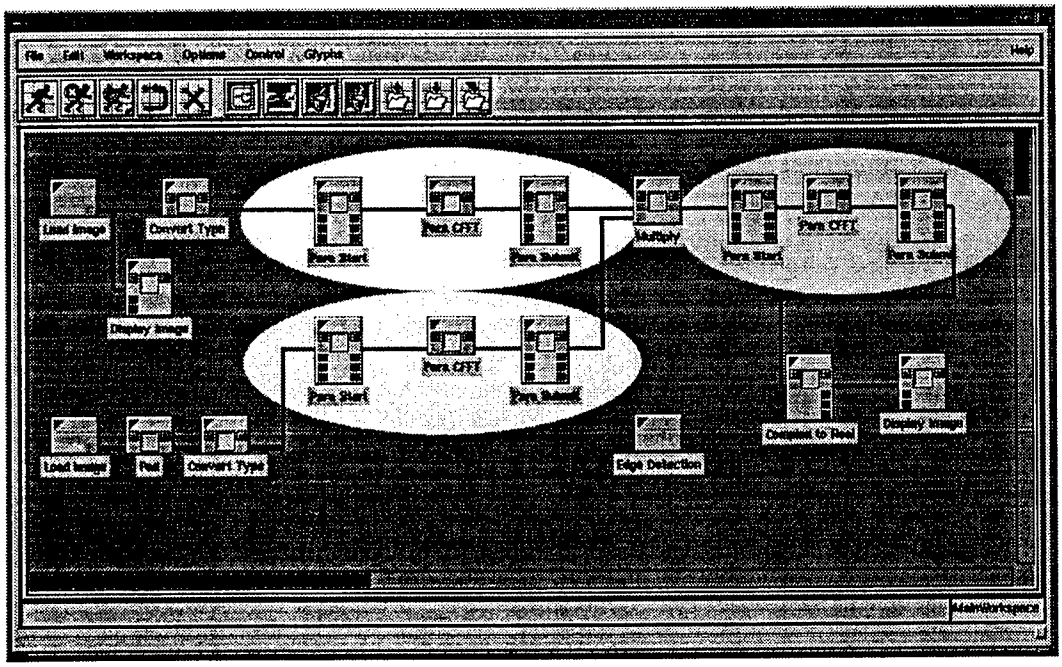
Software abstractions achieve platform independence

- Abstraction layer allows system to support multiple platforms
- Parallel jobs may be executed
  - sequentially
  - simultaneously
  - simultaneously using multiple platforms
- Currently supported parallel operating environments and platforms
  - IBM SP batch (local and remote)
  - IBM SP interactive
  - Cluster of workstations
- Other POEs and platforms can be added by cloning existing shell scripts and tailoring them to the local environment

Other features of RLSTAP_HPC

- The RLSTAP_HPC parallelization tools can be applied to any number of algorithm problems both in and out of the RLSTAP environment
- Parallel scientific libraries can be utilized in conjunction with RLSTAP_HPC software tools to perform algorithm functions
- Supported data distribution modes include those supported by Khoros Parallel Data Services
- Additional data distributions can be performed within parallel routines using direct MPI calls provided they are gathered to a state that Khoros PDS expects before exit
  - Commonly occurs when utilizing parallel scientific libraries
  - Can be employed to perform distributions in higher dimensions

Parallel edge detection workspace



Parallel FFT performance

Timings for the FFT in MPI.

2D Complex to Complex FFT
with PESSL routine pdcft2 on the Maui/SP2 - wide nodes

Size of array / # procrs	1	2	4	8	16
64 x 64	2.33e-3	2.64e-3	1.93e-3	1.65e-3	2.52e-3
128 x 128	1.14e-2	9.24e-3	5.07e-3	3.59e-3	3.35e-3
256 x 256	4.86e-2	3.99e-2	2.05e-2	1.18e-2	8.05e-3
512 x 512	2.36e-1	1.74e-1	8.44e-2	4.66e-2	2.76e-2
1024 x 1024	9.20e-1	7.30e-1	3.48e-1	1.83e-1	1.06e-1

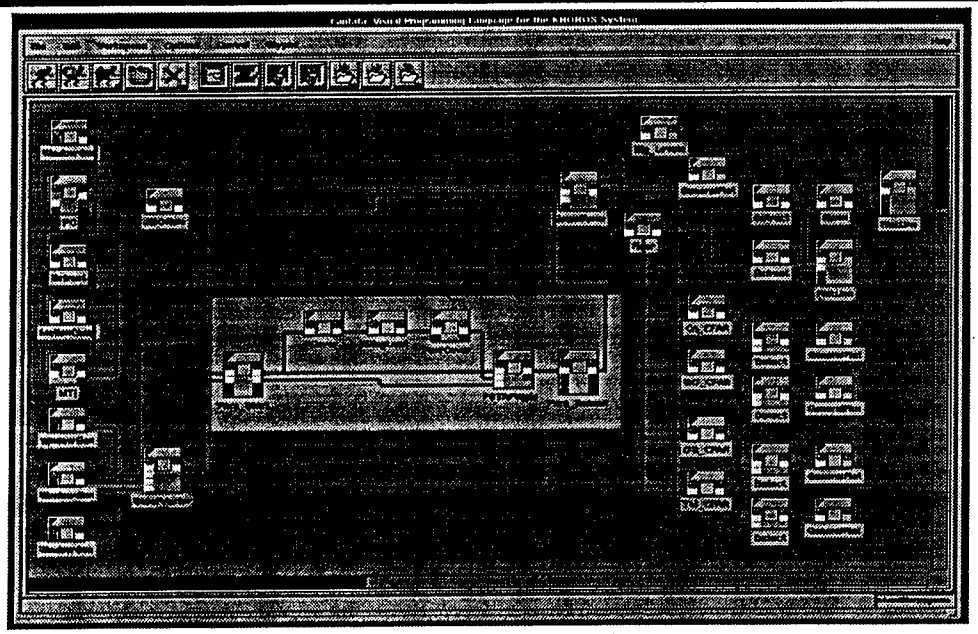
RLSTAP parallelization example: Inverse covariance

- Inverse covariance calculation is utilized in STAP algorithms
- Two methods of calculation
  - Gaussian elimination
  - Complex singular value decomposition
- Parallelization can be achieved in two ways
  - fine-grain parallelism in which a single matrix is distributed among multiple processors
  - coarse-grain parallelism in which different matrices are operated upon by each processor
- Two parallelization examples are presented here
  - fine-grain inverse covariance using Gaussian elimination

Parallel image processing example: Edge detection

- Executes from standard Khoros environment in Cantata
- Utilizes parallel FFT glyph
  - Can perform 2 or 3 dimensional complex FFT
  - Implemented using calls to IBM's Parallel Engineering & Scientific Subroutine Library (PESSL)

RLSTAP workspace with parallel inverse covariance



Summary

- RLSTAP_HPC provides access to massively parallel computing resources from within the Khoros graphical programming environment Cantata
- It allows the creation of simple-to-use parallel glyphs that execute within the standard Khoros environment
- The IBM SP environment at MHPCC is the target platform but RLSTAP_HPC can also be executed on clusters of workstations
- Parallel scientific libraries can be employed to perform algorithm functions
- RLSTAP_HPC has been demonstrated on both RLSTAP algorithm components and generic parallel image processing problems

8. Para_tools Manual
Intro, Design, Instructions

Joe Fogler
Albuquerque High Performance Computing Center (AHPCC)

30 October, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

Chapter 1 - Introduction

A. Overview

The para_tools toolbox is a collection of tools and libraries to provide an interface from serial Khoros and Cantata to parallel programs running on specialized hardware such as the IBM SP. Also included in the toolbox are demonstrations of tool usage and techniques to be used when creating parallel Khoros jobs.

A.1. Objectives

The main purpose of this toolbox is to demonstrate an interim solution that allows parallel processing using Khoros and the Khoros Cantata programming paradigm. Khoral Research has announced plans for a parallel programming environment in Khoros, but this toolbox is for those who want to do parallel programming in Khoros now.

Sub-goals include:

- Ability to run on multiple parallel platforms
- Executables to be as simple to use as a normal Khoros Cantata glyph
- Anticipate and simplify software migration to future version of Khoros

A.1.1. Solution

The solution, contained in this toolbox, is RLSTAP HPC. Parallel processing is supported using standard Message Passing Interface (MPI) programming on a variety of parallel platforms. The demonstrations in this toolbox are all RS/6000 based, but others can be supported.

To allow parallel processing in Cantata, special parallel glyphs are used. These parallel glyphs may only be executed in the Cantata environment when grouped in a lineup with a Parallel Start glyph in front and a Parallel Submit glyph behind. Parallel Start and Parallel Submit are special purpose glyphs: Parallel Start initializes the parallel environment and Parallel Submit gathers the distributed data and closes the parallel environment.

In keeping with the Cantata paradigm each parallel glyph represents an executable with data "pipes" between glyphs. So, for each parallel glyph there is a parallel executable that will be executed on the parallel machine. All inputs and outputs for the parallel glyphs must pass through Parallel Start and Parallel Submit, i.e., no inputs can be connected to a parallel glyph that have not come out of Parallel Start, and no outputs from a parallel glyph can be used until it is put into Parallel Submit. This allows all parallel data to be synchronized on the parallel machine. (There is nothing physically stopping someone from bypassing Parallel Start or Parallel Submit and wiring an input directly into a parallel glyph, but the behavior of the glyph would be undefined

and it would almost certainly generate an error and quit. The output of a parallel glyph does not contain useful information to the user and there is, therefore, no reason not to pipe it through Parallel Submit.)

This system allows multiple platforms to be used simply by changing the configuration of Parallel Start. As we shall see later, this even allows multiple platforms to execute different portions of the same workspace simultaneously. Implied is the fact that the parallel hardware need not (and, in fact, frequently is not) the same as the serial hardware running Cantata.

With the addition of Parallel Start and Parallel Submit the use of parallel glyphs is exactly the same as for standard serial glyphs.

Parallel programs that can be integrated into Cantata using these tools and techniques are ordinary parallel codes: All examples here use MPI and C but they could, be extended to PVM or other programming languages such as FORTRAN 90 or HPF.

The demonstrations in this toolbox show examples running Khoros on AIX workstations and the parallel code on the IBM SP both interactively and batch, and on a cluster of RS6000 workstations.

Chapter 2 - Design

A. System Design

RLSTAP_HPC was designed with the following features:

- The Cantata workstation is independent of the parallel platform.
- All parallel code and structures is platform independent.
- Khoros data format is used throughout. This means there is no data conversion required between the serial and parallel environments.
- The parallel glyph behavior follows the Khoros paradigm (e.g., the inputs and outputs are defined in a standard Khoros manner, the idea of building blocks is incorporated, etc.).

In order to facilitate several of the above features a layered structure was adopted. This structure can be viewed from two different perspectives: The horizontal and the vertical. Horizontally is the program flow through a set of parallel glyphs. Vertically is the five layer structure used to implement the program flow.

A.1. Parallel Program Flow

In a normal Cantata program control and data flows through "pipes" from one glyph to the next. A glyph does not execute until all of its inputs are ready, and enables its output(s) when it finishes.

In a parallel glyph lineup (all the glyphs from the Parallel Start to the Parallel submit, inclusive) only control and some administrative data flows through the pipe. Associated with each parallel glyph is the normal Khoros serial executable and a new parallel executable, referred to as an "agent". The agent is activated by the serial executable via the agent's command line interface. Data flows directly from one agent to the next.

So, for example, the Parallel Start glyph splits the control and data flow. The first parallel glyph (the one immediately following Parallel Start) receives the control flow, while its associated agent receives the data flow. When Cantata activates the parallel glyph the glyph uses the command line interface to "call" the agent. When the parallel computation is completed data is passed to the input of the next agent, and control is passed (via Cantata) to the next parallel glyph. The control and data streams are rejoined by the Parallel Submit glyph

This simple model is basically complete, but there are a few more details. Each agent checks the data distribution of its input against the distribution it requires. If they are different, it redistributes the data. Using this mechanism the data is actually distributed by the agent associated with the first parallel glyph in the lineup, not by Parallel Start. When the data stream is rejoined to the control stream, it is actually done by an agent

associated with Parallel Submit which gathers the data back from its distributed state. The implementation of this model is quite a bit more complex (surprise!). Which leads us to the vertical model, layers.

A.2. Layers

The implementation of a parallel executable, from Cantata glyph to raw MPI code consists of five layers:

1. Kroutine
2. Abstract layer
3. Pkroutine
4. Lkroutine
5. Lroutine

A.2.1. Kroutine

The main difference between a serial Kroutine and a parallel Kroutine is where the data is. In the serial scenario the data to be manipulated is passed into and out of the program through the input and output parameters on the command line. In the parallel case these inputs and outputs correspond to administrative data about the parallel process. This is possible because the parallel data is being passed between modules at a lower level (and, as a result, being passed on the parallel machine, not the controlling serial machine).

Parallel Kroutines are normal Khoros Kroutines. Their job is to accept input, do any processing on it that may be necessary, pass the data to the Abstract layer, process the results from the Abstract layer, and pass on the output.

The structure of a parallel Kroutine is as follows.

- Normal housekeeping chores. These include allocating memory, declaring variables, etc.
- Read control information from each input. This control information is in the form of a C structure and includes
  - currentProcessNum -- incremental count of how many parallel glyphs have been processed.
  - nNodes -- the number of nodes the current parallel job uses.
  - stdo -- flag indicating if standard output should be displayed by each glyph.

- timeStamp -- a string containing the time and date the job started. This is used to sync the timing of parallel threads in the job.
- initialDir -- initial directory where the parallel job is started.
- jobName -- name of this job (theoretically unique)
- class -- job class. This is typically used on batch SP nodes to specify the queue the job will run in.
- notification -- a string indicating the user notification action(s) to be taken. Used mostly by the SP batch processor.
- user -- who started this job?
- display -- string where the DISPLAY variable is stored. Used for batch mode.
- inDataFileName -- the name of the actual input file used by the parallel routine for this input.
- outDataFileName -- the name of the actual output file used by the parallel routine for this output. This value is not actually read in, but generated just before it is used.

- Prepare and call the Abstract layer. Two items are always passed to the Abstract layer: the job name (obtained from jobName, above) and the parallel job binary directory (obtained from the environment). These items allow the Abstract layer to find more information it may require that was stored during the execution of the Parallel Start glyph. The Kroutine then supplies the command line of the parallel job that must be run. This allows the Abstract layer to do whatever is necessary to prepare a particular parallel environment, then execute the command line passed to it.

  A mechanism is implemented which allows the Kroutine to tell the Abstract layer that a temporary directory is to be used. If the substring "@localscratch" is included in a path name or a command line string it is automatically converted by the Abstract layer to a path local to the parallel nodes being used.

- Output control data. The control data for the next parallel operation is updated (the input file is changed to the current output file, etc.) and written to the output file of the Kroutine.

- Cleanup. Normal cleanup items like free memory close files, etc.

A.2.2. Abstract layer

The Abstract layer provides a mechanism for simply and quickly changing and/or adding machine specific capabilities to the parallel execution environment. As such it is an interface between the serial Kroutine and the parallel kroutine, the Pkroutine. This layer is highly platform dependent on the parallel architecture being used. Included in this toolbox are several examples of script sets that implement the abstract layer of the job

submission process:

- para_tools/bin/IBM_SP_Batch submits a job to the IBM LoadLeveler for batch processing, and waits for the job to complete before returning. Each parallel glyph ends up creating an entry in a LoadLeveler script which is then submitted by Parallel Submit.

- para_tools/bin/IBM_SP_Interactive allows a job to be run on a set of interactive nodes using interactive POE. Each glyph ends up running the parallel code immediately, leaving Parallel Submit to merely gather the distributed data.

- para_tools/bin/CIRT_SP2 is a quick-and-dirty implementation of a remote machine batch submission. Using remote shells and remote copies the data is copied to the remote machine, the job submitted, and, on completion, the output data is copied back.

- para_tools/bin/RS6000_Workstations is an example of running on a cluster of workstations. Unlike the SP interactive example this implementation behaves more like the batch processes, gathering command lines in a script file and allowing Parallel Submit to run them all.

These abstract layer directories show up as options in Parallel Start to specify the parallel platform to be used.

NOTE: These directories hang off the toolbox /bin directory but are not included in the toolbox make system. Any changes or new abstract layers must be constructed entirely by hand.

Due to the vast differences between two installations of a machine type, not to mention the difference between the installation of two machines of completely different type, the abstract layer must be customized for each individual machine and site. The recommended method is to copy one of the examples provided in this toolbox and modify it, rather than starting from scratch.

There are three main scripts (or programs, if the implementer so chooses) to perform all functions:

- para_start.cmd -- This script accepts all job information gathered by the Parallel Start glyph and starts the processes of building the parallel job. This includes dumping most of the information from the command line into a temporary file so that subsequent scripts can find it. This allows information such as the number of nodes, queue, etc. to be passed in by Parallel Start and ignored by the other glyphs in the parallel lineup.

  This script will also open any batch control files required, etc.

- para_op.cmd -- This script is the one called by all the parallel glyphs with the exception of Parallel Start. On the command line it expects to see the job name, by which it identifies the temporary file created by para_start.cmd, the directory where the parallel job binaries can be found, followed by the command line to be executed on the parallel machine. This command line does NOT include parallel execution information such as "poe", but just a command line consisting of the executable path/name and the command parameters. The script then does whatever is necessary to execute the command line on each node.

For a batch job, for instance, para_op.cmd might put

"poe command_line"

in a batch control file, which will be executed by para_submit.cmd. For an interactive job in might just execute the above line directly.

para_op.cmd is also called by Parallel Submit to perform the data gather, since gather is a parallel routine just like any other.

- para_submit.cmd -- This script is called by the Parallel Submit glyph. It does whatever is required to wrap up the parallel execution. In a batch environment, for instance, it might submit the batch control file to the batch scheduler and wait for a response. In an interactive environment it might only have to delete the temporary file created by para_start.cmd and do any other cleanup required.

Any other files, programs, or scripts which may be required by these three scripts can also be included in the same directory. In this toolbox the IBM_SP_Batch directory contains not only the three required scripts, but scripts to submit batch jobs to LoadLeveler, wait for their return, and check the status of submitted jobs.

As can be seen the manner in which the parallel job appears to execute (to the user sitting in front of a Cantata screen) can vary greatly. An interactive system may take a considerable amount of time for every glyph in the parallel lineup, while a batch environment might whiz through all of the glyphs until Parallel Submit is encountered, then seem to freeze for a long period of time as the batch job is submitted, queued, executed, and returned.

A.2.3. Pkroutine

The Pkroutine is the first layer to be executed on the parallel machine. Its function is to provide a place-holder for future releases of Khoros, where this layer will be incorporated into Khoros itself. In some ways it resembles a Khoros kroutine: the major difference is that it does not contain a pane (a pane would be a pain to reference from the parallel machine). It opens the data input file, passes that data to the Lkroutine, gets data back from he Lkroutine, and writes it to the output. It is not recommended that anything else be done at this level, as it will go away in future releases of Khoros.

A.2.4. Lkroutine

The primary responsibility of the Lkroutine layer is to deal with Khoros. All data in the system uses Khoros KDF file format. In order to ease conversion of existing MPI codes into parallel Khoros codes the Lkroutine layer takes responsibility for converting between KDF format and a raw data format more palatable to an MPI program. This layer is also assigned the responsibility for data distribution.

The Lkroutine receives the data from the Pkroutine in KDF format. It then checks the distribution of the data: if it is correct the data is passed to the Lroutine. If the data is not distributed, or is distributed incorrectly, the Lkroutine (re)distributes it appropriately, then passes it to the Lroutine. Likewise, when the Lroutine returns

data the Lkroutine converts it back to KDF format and passes it back up to the Pkroutine.

The Lkroutine must be coded with the distribution requirements of the Lroutine. If the Lroutine can accept multiple input distributions, the Lroutine should also accept those distributions. If the Lroutine expects a distribution not supported by Khoros Data Distribution Services, the correct distribution should be created in the Lkroutine using standard MPI calls. Likewise the output should be labeled with the correct current distribution, which is whatever distribution the Lroutine hands the data back in.

The idea here is to allow the use of existing MPI codes by merely stripping the distribution details out (letting the Lroutine handle that) and calling it from the Lroutine. Minimal conversion effort.

A.2.5. Lroutine

The Lroutine is a raw MPI code. It knows nothing of Khoros data objects. It can assume the data is already distributed in an acceptable format. This layer uses standard MPI calls to manipulate data. A library call may take the place of the Lroutine (essentially the library becomes the Lroutine), as was done in the cFFT glyph in this toolbox. The essence here is that the Lroutine is a pure MPI program, with no knowledge of Khoros or the Cantata environment.

A.3. Parallel glyphs

There are three types of parallel glyphs:

1. Parallel Start
2. Parallel Operation
3. Parallel Submit

All three types of glyphs are required in a parallel lineup.

A.3.1. Parallel Start

The Parallel Start type is both a type and a glyph. The only glyph of this type is the Parallel Start glyph itself. The purpose of the Parallel start glyph, from the users perspective, is to delineate the beginning of a parallel lineup.

Most of the global parameters required by the parallel job are specified in the pane of Parallel Start. Among the items that can be specified in the pane are:

- Input and output filenames

- Standard output suppression -- suppress standard output from all glyphs in the parallel lineup
- Parallel environment -- the parallel environment to use when executing the lineup
- Initial Directory -- directory where jobs are executed/submitted
- Job Name -- a name for the parallel lineup that must be unique on the workspace
- Number of parallel nodes
- Class -- queue name for batch jobs
- Notify user after batch completion

From the programmer's perspective Parallel Start allows the system to prepare whatever resources might be required by subsequent Parallel Operation glyphs. The only real invariant here is that, at the abstract layer, Parallel Start must write a temporary file so Parallel Operation and Parallel Submit glyphs can get information specified in the Parallel Start glyph.

Currently every condition that applies to any parallel environment is included on the Parallel Start pane. This means that, although data is gathered for all fields, and passed to the abstract layer, all of the data may not be relevant to the environment selected. For example the Class field does not apply to interactive execution, since it refers to the batch queue to be used. While this "shotgun" approach may be inefficient, it is easy to code and maintain.

One thing the programmer must look out for when updating the pane is the "Environment" field. This field shows the name of a directory where the abstract layer command files can be found. This is hard-wired in the field, and in the toolbox. There is no enforcement to make sure they agree. Make sure the environments listed on the pane match those in the $TOOLBOX/bin directory. Similarly the Class field and Notification field are mated to code in the abstract layer that must be matched by hand.

All inputs from the parallel lineup that come from "outside," i.e., from serial glyphs, must pass through the Parallel Start glyph. Currently the Parallel Start glyph supports four inputs and outputs. This number is arbitrary, and is easily changed in the Parallel Start Kroutine.

A.3.2. Parallel Operation

All glyphs that perform some sort of parallel calculation are of the Parallel Operation type. Examples in this toolbox include Parallel cFFT and Parallel Multiply. Operations of this type take distributed data as input, manipulate the data, and output distributed data. Each Parallel Operation redistributes the data if it is not already in an acceptable format (hence the first Parallel Operation in a lineup will take undistributed data and distribute it as it pleases). If the user is not careful, this could result in a redistribution between every glyph in the lineup.

A.3.3. Parallel Submit

Parallel Submit, like Parallel Start, has only one glyph in the type: The Parallel Submit glyph. Parallel Submit (so named because, in a batch system, this is where the job finally gets submitted) has two duties: to gather the distributed data so that it may be passed to subsequent serial glyphs, and clean up the parallel environment. An example of the kind of cleanup Parallel Submit should do is deleting the temporary file Parallel Start created on the abstract layer. Also, in a batch environment this is likely where the batch job is actually submitted.

All outputs from the parallel lineup that go "outside" to serial glyphs must pass through the Parallel Submit glyph. Currently the Parallel Submit glyph supports four inputs and outputs. This number is arbitrary, and is easily changed in the Parallel Submit Kroutine.

A.4. Using the parallel glyphs

In order to execute algorithms using the parallel glyphs the user must do the following things.

1. Place a Parallel Start and a Parallel Submit glyph on the workspace.

2. Select the parallel algorithm glyphs you want to use and place them between the Parallel Start and Parallel Submit glyphs on the workspace. NOTE: While you can place normal serial Khoros glyphs in between the Parallel Start and Parallel Submit glyphs, and even hook them up, they will NOT work properly. They will try to use the administrative data being passed between parallel glyphs, which is not the same as the parallel data. Do not do this! Likewise, parallel glyphs do not work outside a parallel lineup bracketed by Parallel Start and Parallel Submit.

3. Wire all inputs from the serial world to the inputs of the Parallel Start glyph.

4. Wire the corresponding outputs of Parallel Start to the parallel glyphs which require the input.

5. Interconnect the parallel glyphs in the normal manner.

6. Wire all outputs to the serial world to the inputs of Parallel Submit.

7. Wire the corresponding outputs of Parallel Submit to the appropriate serial glyphs.

8. Open the Parallel Start pane and select a parallel environment, and fill in the appropriate information for that environment.

9. Be sure to set the parallel glyph panes appropriately.

10. Run the lineup.

B. Object Manifest

Parallel Tools toolbox Object Manifest			
Category Subcategory	**Operator**	**Description**	**Executable**
		2d wiener filter	`pkwiener`
dev			`pkgather`
support		scatter routine	`pkscatter`
para_tools 2D Filters	Wiener 2D Filter	Wiener filter. Expects 2D FFTed data.	`Wiener_2D_filter`
para_tools	Para Multiply	Quick and dirty pointwise multiply	`Para_Multiply`
Arithmetic	pkMultiply	Quick and dirty pointwise multiply (parallel portion)	`pkMultiply`
para_tools	Para Start	Parallel start module	`Parallel_Start`
Required	Para Submit	Parallel submission module	`Parallel_Submit`
para_tools		Parallel complex FFT (2D/3D)	`pkcfft`
Transform	Para CFFT	Parallel Complex FFT, 2 or 3D	`cFFT`
para_tools support	Scatter	Scatter a KDF object using DDS	`Scatter`

Chapter 3 - Instructions

A. How to build the abstract layer

The abstract layer is the interface between the Khoros Cantata glyphs and the individual parallel environments. While the glyphs can specify high-level things like the number of processors and the like, the abstract layer is where those abstractions are translated into commands and programs to execute on a particular parallel environment. This section will explain how the abstract layer does what it does, and how to construct one for a new parallel architecture.

A.1. How the abstract layer works

There are three major components to the abstract layer:

- para_start.cmd
- para_op.cmd
- para_submit.cmd

para_start.cmd and para_submit.cmd correspond directly to the Parallel Start and Parallel Submit glyphs, respectively.

The Parallel Start glyph calls para_start.cmd, passing all of the information gathered in the Parallel Start pane. Among other things this information contains the job name for this parallel lineup. Using this job name (and, depending on the parallel architecture, the user's UID and/or PID) a temporary file is created which contains all the information received from the Parallel Start pane that is pertinent to this particular parallel architecture. In this way the various components of the abstract layer have access the the pane information from the Parallel Start glyph, without the parallel glyphs having to tote this information around and download it to the abstract layer each time. Only the job name is needed to find this temporary file. After the temporary file is built the para_start.cmd program must do whatever is necessary to set up the parallel environment.

The parallel glyphs (such as a parallel FFT glyph) all call para_op.cmd. para_op.cmd uses the data stored in the temporary file created by para_start.cmd, along with the command line to run the Pkroutine supplied by the parallel glyph, and creates a command line that will execute the Pkroutine on the specific parallel architecture. This command line can be stored to a file, as would happen in a batch system, or can be executed immediately, in an interactive parallel environment, for instance.

The Parallel Submit glyph calls para_submit.cmd. All of the pane information is passed, but that only consists of the input and output file names. para_submit.cmd does whatever cleanup is necessary to the parallel environment. This includes deleting any temporary files created by para_start.cmd and para_op.cmd. para_submit.cmd might also submit a command file to the scheduler in a batch environment.

A.2. The individual components

In this section we will use, as an example, the IBM SP batch abstract layer provided in this toolbox ($TOOLBOX/bin/IBM_SP_Batch). This abstract layer provides complete services to IBM LoadLeveler, a batch scheduler used on the IBM SP.

A.2.1. para_start.cmd

```
#!/bin/ksh
#
# para_start.cmd
#
# Set up a LoadLeveler script in preparation to submit to an IBM SP.
# Two scripts are created... a LoadLeveler script and a command file.  The
# LoadLeveler script has the LoadLeveler parameters in it (some determined
# by command line inputs, some hard-wired).
#

#
# Globals
#
mkdir="/usr/bin/mkdir"

#
# Check to make sure $TMPDIR != /usr/tmp:  That isn't local on the SP's
#
if [[ $TMPDIR = "/usr/tmp" ]]
then
 print -u2 'ERROR:  $TMPDIR must NOT be defined to be /usr/tmp'
 exit 1
fi

#
# Parse the command line for the arguments we need.  Extraneous args discarded.
#

while [[ $1 != "" ]]
do
 arg=$1
 shift
 if [[ $arg = "-initialDir" ]]
 then
  initialDir=$1
 elif [[ $arg = "-jobName" ]]
 then
  jobName=$1
 elif [[ $arg = "-nNodes" ]]
 then
  nNodes=$1
```

```
  elif [[ $arg = "-class" ]]
  then
    class=$1
  elif [[ $arg = "-notifyUser" ]]
  then
    notifyUser=$1
  elif [[ $arg = "-notification" ]]
  then
    notification=$1
  elif [[ $arg = "-parJobBinDir" ]]
  then
    parJobBinDir=$1
  fi
done

#
# class and notification come back as numbers and must be converted to text.
#
# ********** NOTE: This requires MANUAL coordination with the
# **********       para_start pain.

case $class in
  1)  class=short;;
  2)  class=mixed;;
  3)  class=long;;
  4)  class=bigmem;;
  5)  class=large;;
  6)  class=medium;;
  7)  class=small_long;;
  8)  class=small_short;;
  *)  class=short;;  # make short the default
esac
case $notification in
  1)  notification=always;;
  2)  notification=complete;;
  3)  notification=error;;
  4)  notification=never;;
  5)  notification=start;;
  *)  notification=never;;  # make never the default
esac

#
# Store all processed input EXCEPT the jobName in the file $jobName.abs.tmp.
# This is where subsiquent modules will go to find this information out.  It
# is deleted by the para_submit.cmd script.
#

tmpfile="$TMPDIR/$jobName.abs.tmp"

print -- -initialDir $initialDir                    > $tmpfile
```

```
print -- -jobName $jobName                    >> $tmpfile
print -- -nNodes $nNodes                      >> $tmpfile
print -- -class $class                        >> $tmpfile
print -- -notifyUser $notifyUser              >> $tmpfile
print -- -notification $notification          >> $tmpfile

#
# Build the LoadLeveler file.  The filename is $jobName.ll (e.g., pk_job1.ll)
#

llfile="$initialDir/$jobName.ll"
cmdfile="$initialDir/$jobName.cmds"
#print $llfile

print "#"                                     > $llfile
print "# Script initially generated from Para Start"    >> $llfile
print "#"                                     >> $llfile
print "# Ensure this is C-shell script"       >> $llfile
print "#!/bin/csh"                            >> $llfile
print "#"                                     >> $llfile
print "# Job Command Keyword Initialization"  >> $llfile
print "#@ shell = /bin/csh"                   >> $llfile
print "#@ cpu_limit = 600"                    >> $llfile
print "#@ wall_clock_limit = 600"             >> $llfile
print '#@ requirements = (Adapter == "hps_user")'    >> $llfile
print "#@ checkpoint = no"                    >> $llfile
print "#@ restart = no"                       >> $llfile
print "#@ job_type = parallel"                >> $llfile
print "#@ job_name = $jobName"                >> $llfile
print '#@ output = $(job_name).$(Cluster).out'    >> $llfile
print '#@ error = $(job_name).$(Cluster).err'     >> $llfile
print "#@ initialdir = $initialDir"           >> $llfile
print "#@ min_processors = $nNodes"           >> $llfile
print "#@ max_processors = $nNodes"           >> $llfile
print "#@ class = $class"                     >> $llfile
print "#@ notification = $notification"       >> $llfile
print "#@ notify_user = $notifyUser"          >> $llfile
print "#@ group = asp"                        >> $llfile

#
# Build environment variables
#

environment="environment ="
environment="$environment MP_INFOLEVEL = 2"
environment="$environment;MP_LABELIO = yes"
environment="$environment;MP_STDOUTMODE = ordered"
environment="$environment;MP_EUILIB = us"
environment="$environment;MP_RESD = yes"
environment="$environment;MP_EUIDEVICE = css0"
```

```
environment="$environment;MP_RMPOOL = 0"
environment="$environment;MP_HOSTFILE = NULL"
environment="$environment;MP_PULSE = 0"
environment="$environment;DISPLAY = $DISPLAY"
environment="$environment;MP_NEWJOB = yes"
environment="$environment;"

print "#@ $environment"                          >> $llfile
print "#@ queue"                                 >> $llfile
print "#"                                        >> $llfile
print "poe -cmdfile $initialDir/$jobName.cmds"   >> $llfile

#
# Make the temporary directory for intermediate data on the nodes
#

rm -f $cmdfile > /dev/null 2> /dev/null
print $mkdir /localscratch/asp_$jobName          >> $cmdfile
```

The first thing the script does is to check for invalid values for the environment variable TMPDIR. Since on the IBM_SP /usr/tmp is often shared among the nodes it will not do as a temporary directory. The program dies if this is the case, as there is no point in doing any more calculation.

Next the program parses the command line looking for arguments, which are as follows. Except as noted all values are obtained from the Parallel Start pane.

- initialDir -- The directory given to LoadLeveler as the initial directory.
- jobName -- This name is used to uniquely identify a parallel job and is used in the construction of temporary file names and the like. This is one of two parameters passed by ALL parallel glyphs.
- nNodes -- The number of nodes to be requested.
- class -- The queue in which LoadLeveler is to place the job.
- notifyUser -- The email address of the user to be notified.
- notification -- A flag telling LoadLeveler to notify or not notify the user by email.
- parJobBinDir -- The binary directory where the parallel executables can be found. This is the second parameter passed by every parallel glyph. It is not taken from the Parallel Start pane, but is in fact the binary directory of the parallel glyph initiating the call.

Next is a modification of the values received for class and notification. Since Khoros returns the number of the list box selected, instead of the label, this conversion re-associates the name with the variable. Notice that there is no automatic method for keeping this straight... it is strictly a manual process.

The values of the command line parameters are placed in a temporary file where the other abstract layer programs can find it, in this case in the current temporary directory with a filename constructed from the unique job name (which is the same for all parallel glyphs in a lineup).

Now the script builds the LoadLeveler command files. These files consist of a LoadLeveler script, and a POE command script. In this way several executables can be run sequentially without reallocating nodes and resources between each one. Most of the LoadLeveler parameters are hard-wired, with the values received on the command line substituted in. A more flexible system would ask the user more questions, and hard-wire less of the parameters. One critical parameter that is hardwired in this example is the time limits. These probably would be variables set on the Parallel Start pane.

Most of the environment is also hard-wired. This reduces the risk of user error (very easy with these variables) but limits flexibility.

The last line of the LoadLeveler script is the invocation of POE, with the POE command file as input. The only line in the POE command file at this point is a command to make a temporary directory on each node.

So, when this script is finished running, there is a complete LoadLeveler script, ready for submission, a POE command file with the first parallel command in it, and a temporary file with the parameters extracted from the Parallel Start pane in it for downstream processes to reference.

A.2.2. para_op.cmd

```
#!/bin/ksh
#
# para_op.cmd
#
# Append the input command to the command file. This script allows any of the
# inputs/outputs to contain the string "@localscratch". The string
# "/localscratch/asp_XXX" is substituted for that string, where XXX is the
# jobname. NOTE: This means that if a jobname is not supplied _BEFORE_ the
# "/localscratch" string on the command line, this substitution will NOT work
# correctly!
#

#echo $*

#
# Parse the command line for the arguments we need. Extraneous args discarded.
#

jobName=$1
shift
parJobBinDir=$1
shift
cmdOp=`echo $* | sed -e s?@localscratch?/localscratch/asp_$jobName?g`

#
# Dig through the temporary file created by para_start.cmd for rest of the
```

```
# arguments we need.
#

exec 3< $TMPDIR/$jobName.abs.tmp
read -u3 var value
while [[ $var != "" ]]
do
 if [[ $var = "-initialDir" ]]
 then
  initialDir=$value
 fi
 read -u3 var value
done
exec 3<&-

#
# Append a command to the command file.  The filename is $jobName.cmds
# (e.g., pk_job1.cmds)
#

cmdfile="$initialDir/$jobName.cmds" #print $cmdfile

print time $parJobBinDir/$cmdOp  >> $cmdfile
```

In the batch environment para_op.cmd has a very simple job. First it extracts the job name and the parallel job binary directory from the command line. Everything else on the command line is assumed to be the parallel executable name and its parameters.

In these scripts a simple mechanism is implemented to know when to assign a file to temporary local storage on a node. When a file, lets say a temporary data file that is passed from one parallel glyph to the next, needs to be stored in local temporary storage the value "@localscratch" is embedded in the path. The sed operation in this script removes that value and replaces it with a temporary directory local to each node. In the case of the IBM SP at Maui that directory is "/localscratch".

Next the script loads the environment variables it will need from the temporary file created by para_start.cmd. In this case only the initial directory is required. Now the script can do what it is here for: open the POE command file and append a line to execute the parallel command passed to it from the command line (with the appropriate modifications).

A.2.3. para_submit.cmd

```
#!/bin/ksh
#
# para_submit.cmd
#
# Append to the command file to gather the results and submit the LoadLeveler
# script.  This script allows any of the inputs to contain the string
# "@localscratch".  The string "/localscratch/asp_XXXX" is substituted for
# that string, where XXXX is the jobname.  NOTE:  This means that if a
```

```
# jobname is not supplied _BEFORE_ the "/localscratch" string on the command
# line, this substitution will NOT work correctly!
#

#echo $*

#
# Globals
#
rm="/usr/bin/rm"

#
# Parse the command line for the arguments we need.  Extraneous args discarded.
#

while [[ $1 != "" ]]
do
 flag=$1
 shift

 # Replace "@localscratch" with "/localscratch"
 if [[ $1 = @localscratch* ]]
 then
  arg="/localscratch/asp_$jobName"${1#@localscratch}
 else
  arg=$1
 fi

 if [[ $flag = "-jobName" ]]
 then
  jobName=$arg
 elif [[ $flag = "-binDir" ]]
 then
  parJobBinDir=$arg
 elif [[ $flag = "-initialDir" ]]
 then
  initialDir=$arg
 fi
done

#
# Append a command to the command file.  The filename is $jobName.cmds
# (e.g., pk_job1.cmds)
#

llfile="$initialDir/$jobName.ll"
cmdfile="$initialDir/$jobName.cmds"

print "$rm -rf /localscratch/asp_$jobName"              >> $cmdfile
```

```
#
# Submit the job to LoadLeveler
#
jobID=`$parJobBinDir/IBM_SP_Batch/llsubwait $llfile $parJobBinDir/IBM_SP_Batch`

#
# Erase the temporary information file created by para_start.cmd.
#
rm -f "$TMPDIR/$jobName.abs.tmp" > /dev/null 2> /dev/null

#
# Copy the parallel output to stdout
#
cat $initialDir/$jobName.${jobID##*.}.out
```

In the batch system, para_submit.cmd does a great deal of the work. First the command line is parsed for input variables. The temporary directory substitution is made on each parameter, if necessary. The parameters are

- jobName -- The unique job name of the lineup.
- parJobBinDir -- The directory where the parallel binary to be executed can be found (in this case where the routines to submit a LoadLeveler job can be found).
- initialDir -- The directory where the LoadLeveler script and POE command file are stored.

A command is added to the POE command file to remove the temporary directory on each node that was created by para_start.cmd. Then the job is submitted to the LoadLeveler via another script called "llsubwait". We won't discuss how llsubwait works here: suffice it to say that it submits the job to LoadLeveler, then waits for it to complete before returning. If you wish to know more the script itself can be found at $TOOLBOX/bin/IBM_SP_Batch/llsubwait.

Finally the output of the job is copied to standard out. This allows the output to be viewed as a note to the glyph.

B. Instructions for Creation of a Parallel Toolbox

Kroutines and their associated Pkroutines may be created and maintained as regular Khoros objects, with one exception in the case of the Pkroutine which will be explained later. The two objects must be in the same toolbox. Alternatively, a Kroutine may be constructed to launch a parallel routine which is not a Khoros object, as long as the binary or a link to it appears in the bin directory of the Kroutine's toolbox. It is advisable to keep objects destined for parallel execution in a toolbox separate from normal serial objects since modifications will be made to the toolbox configuration files.

The Khoros distributed data services toolboxes, Message and Paraserv, must be installed on the system. The

Para_tools toolbox should be accessible on the system, and a reference to it should appear in the user's .Toolboxes file.

B.1. Modify the Toolbox Configuration Files to Support Parallel Executions

After the user's toolbox has been created via craftsman, the following steps set up access to the Khoros PARASERV toolbox, MPI and any required scientific libraries. The instructions inside brackets <> refer to the sequence of Khoros operations to follow to make the needed change.

1. Edit the TOOLBOX.def file

   <craftsman : Toolbox Attributes : Files : Edit Imake config file>

   This change is required to include distributed data services.

   - replace: '#include DESIGN_INCLUDE'
   - with: '#include PARASERV_INCLUDE'

This change will depend on what (if any) parallel libraries are referenced from the parallel executable. For example if the parallel code uses calls to PESSL routines, library definitions for the system PESSL libraries and any dependencies should appear here.

- add: 'AddLibrary(pessl, C, /usr/lib);'
- add: 'AddLibrary(blacs, C, /usr/lib);'

Also if the home directory of the libraries defined is not on the link path, it may be necessary to add a reference to it, for example:

- add: 'TOOLBOX_LIBDIR +<= -L/usr/local/scalapack'

2. Edit the TOOLBOX.h file

   <craftsman : Toolbox Attributes : Files : Edit toolbox include>

   - replace: '#include <design.h>'
   - with: '#include <paraserv.h>'
   - add: #define NAME_SIZE 1024

- add: #define CMD_SIZE 8196

- add: mpiStruct definition under typedefs

```
typedef struct {
  int currentProcessNum;      /* incremental count of glyph count so far */
  int nNodes;                 /* Number of nodes */
  int stdo;                   /* should standard output be displayed by each glyph (flag) */
  char timeStamp[50];         /* time & date Para Start started */
  char opEnv[500];            /* currently unused */
  char initialDir[500];       /* initial parallel job directory */
  char jobName[50];           /* unique jobname of this job (not enforced!)*/
  char jobEnvName[50];        /* currently unused */
  char class[50];             /* class of parallel job (e.g. short, long, medium) */
  char notification[50];      /* what user notification should occur */
  char user[50];              /* who started this job? */
  char display[100];          /* DISPLAY environment variable */
  char localMachine[50];      /* currently unused */
  char stdoutFileName[500];   /* currently unused */
  char stderrFileName[500];   /* currently unused */
  char inDataFileName[500];   /* parallel data input file name */
  char outDataFileName[500]; /* parallel data output file name */ } mpiStruct;
```

At this point Kroutines and Pkroutines can be added to the toolbox, via the normal Khoros craftsman 'Add Object' menu selection.

B.2. Create Para_kroutine

To create a Kroutine which will launch a parallel job as part of the Para_Start/Para_Submit lineup:

1. Create a new kroutine object, with a pane configured as required for the parameters needed by the parallel executable.

2. Replace the body of the kroutine.c file in the source directory with the Para_kroutine.tmpl template.

3. Edit the kroutine.c file to accommodate parameters specific to this application. The template file contains guidance on this.

B.3. Create Pkroutine

To create the parallel executable which will be launched by the above Kroutine:

1. Create a new kroutine object, and select the 'Do Not Install in Cantata' option.

2. Replace the kroutine.c file with the parallel executable main.c, and add any files for routines called from main.c. Also replace the kroutine.h with an appropriate .h file, and delete usage.c if not used.

3. In the source directory, edit the Imakefile:

   - add 'EXTRA_LOAD_FLAGS = -F/usr/lpp/ppe.poe/lib/poe.cfg:cc' at the bottom, in the customization section

This flag is necessary to run C executables under IBM's Parallel Operating Environment. In general the EXTRA_LOAD_FLAGS variable is a way to force the Khoros Makefile to append things to the link line for your object.

4. Remake the makefiles.

VERY IMPORTANT: Do not use the 'Generate code' button in Composer on this object. This would write over the main.c source file. The one exception to behavior of Pkroutines as compared with the usual behavior of Khoros objects is that in the current release of Khoros a typical kroutine cannot be executed on a remote computer since it will expect to have its pane available. So Pkroutines can be created and maintained by the Khoros Makefile system but should not make any 'kclui' type calls. The kclui_initialize call is replaced by Ghostwriter when the generate code button in Composer is used. This caveat will not be necessary in some future release of Khoros.

B.4. Add Lroutines

Any files with subroutines called from the Pkroutine can be added to the object using the craftsman 'Add file' button. Once the file is present, remake the makefile to add it to the linking process.

C. Templates

The objects 'cFFT' and 'pkcfft' which appear in the Para_tools toolbox have been used as a basis for the templates. The lpkcfft and lcfft routines also provide examples of the use of the Khoros distributed data services calls, and basic setup and use of PESSL routines.

C.1. Template for the Para_kroutine

```
/*
 * Khoros: $Id$
 */

#if !defined(__lint) && !defined(__CODECENTER__)
static char rcsid[] = "Khoros: $Id$";
#endif

/*
 * Copyright (C) 1993 - 1996, Khoral Research, Inc., ("KRI").
 * All rights reserved.  See $BOOTSTRAP/repos/license/License or run klicense.
 */

/* >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> <<<<<<<<<<<<<<<<<<<<<<<<<<<<
  >>>>
  >>>>       Main program for YourProgram
  >>>>
  >>>>  Private:
  >>>>       main
  >>>>
  >>>>  Static:
  >>>>  Public:
  >>>>
  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> <<<<<<<<<<<<<<<<<<<<<<<<<<<< */


#include "YourProgram.h"

clui_info_struct *clui_info = NULL;

/*-------------------------------------------------------
|
| Routine Name: main()
|
|    Purpose: main program for YourProgram
|
|    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|    + This kroutine will be executed by a glyph in a
|    + parallel lineup. Its primary purpose is to generate
|    + the command line for the parallel executable
|    + associated with it. It is best to generate the
|    + kroutine via craftsman and fill in the required
|    + parallel lineup management code via cut/paste.
|    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|
|     Input:
|          char *clui_info->inData_file; {First Input data object}
|          int  clui_info->inData_flag; {TRUE if -inData specified}
```

```
|
|          char *clui_info->outData_file; {Resulting output data object}
|          int  clui_info->outData_flag; {TRUE if -outData specified}
|
|    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|    + Here appears the rest of the clui_info struct built
|    + from the kroutine's pane. All parameters for the
|    + Pkroutine should have pane entries.
|    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|
|     Output:
|    Returns:
|
|  Written By:
|       Date: Jun 26, 1997
| Modifications:
|
------------------------------------------------------------*/

void main(
  int  argc,
  char **argv)
{

     kform  *pane;       /* form tree representing *.pane file */
/* -main_variable_list */

/*
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     + Copy the entire -main_variable_list section as is.
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/
     int ret;
     char *abstractCMD;
     char *parJobBinDir;
     char *nodeOutFile;
     kfile *inDataFile;
     kfile *outDataFile;
     mpiStruct *mpiData;

/* Variables for kerror */
     char *lib = NULL;
     char *rtn = "YourProgram";
/* -main_variable_list_end */

     khoros_initialize(argc, argv, "YOUR_TOOLBOX");
     kexit_handler(YourProgram_free_args, NULL);

/* -main_get_args_call */
     kclui_initialize(PANEPATH, KGEN_KROUTINE,
```

```
            "YOUR_TOOLBOX", "YourProgram",
            YourProgram_usage_additions, YourProgram_get_args,
            YourProgram_free_args);
 /* -main_get_args_call_end */

 /* -main_before_lib_call */
 /*
      ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      + Copy the entire -main_before_lib_call section as is.
      ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 */
 /* Allocate memory */
      abstractCMD = (char *) kmalloc(CMD_SIZE);
      parJobBinDir = (char *) kmalloc(NAME_SIZE);
      nodeOutFile = (char *) kmalloc(NAME_SIZE);
      mpiData = (mpiStruct *) kmalloc(sizeof(mpiStruct));

 /* Initialize file name(s) */
      ksprintf (parJobBinDir,"%sBIN", kprog_get_toolbox());
      ksprintf (parJobBinDir,"%s", getenv(parJobBinDir));

 /* Read database information from input */
      inDataFile = kfopen(clui_info->inData_file, "r");
      ret = kfread(mpiData, sizeof(mpiStruct), 1, inDataFile);
      if (ret != 1)
      {
       kerror(lib, rtn, "Unable to get MPI data from input stream");
       kexit(KEXIT_FAILURE);
      }
      kfclose(inDataFile);
 /* -main_before_lib_call_end */

 /* -main_library_call */
 /*
      ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      + Copy the -main_library_call section. Some
      + modifications to customize for your application are
      + neccessary in this section.
      ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-*/
 /* Setup and call the abstract layer */
      /* Use "@localscratch" to tell the abstraction layer that it is to use
        local scratch disk somewhere */
      ksprintf (mpiData->outDataFileName, "@localscratch%s.out",
            /* use only the name of the outData_file (not the path) */
            strrchr(clui_info->outData_file, '/'));
      ksprintf (abstractCMD, "%s/para_op.cmd ",
            mpiData->opEnv);

      /* Required variables */
```

```
    ksprintf (abstractCMD, "%s %s",
            abstractCMD,
            mpiData->jobName);
    ksprintf (abstractCMD, "%s %s",
            abstractCMD,
            parJobBinDir);
/*
    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    + In this section, the command line specific to your
    + Pkroutine is built. This is a matter of appending
    + the name of the executable and its arguments to a
    + string variable.
    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/

    /* Add our own input variables */

    /* Append the name of the parallel executable */
    ksprintf (abstractCMD, "%s pkYourProgram",
            abstractCMD);

    /* No changes needed here */
    ksprintf (abstractCMD, "%s -i %s",
            abstractCMD,
            mpiData->inDataFileName);
    ksprintf (abstractCMD, "%s -o %s",
            abstractCMD,
            mpiData->outDataFileName);

    /* This is an example of the arguments needed for the fft routine.
      Append the arguments for your pkroutine in a similar fashion. */
    ksprintf (abstractCMD, "%s -f %d",
            abstractCMD,
            clui_info->FFTdir_toggle - 1);
    ksprintf (abstractCMD, "%s -t %d",
            abstractCMD,
            clui_info->trans_toggle - 1);

/*
    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    + No more changes needed after this point. Copy the
    + rest as is.
    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/

    /* Reroute stdout to keep away those annoying "i"s in Cantata */
    if (!mpiData->stdo)
     ksprintf (abstractCMD, "%s > /dev/null",
            abstractCMD);
    /* Call the abstract program */
```

```
        if (ksystem (abstractCMD) != 0)
        {
         kerror (lib, rtn, "Unable to execute command '%s'", abstractCMD);
         kexit(KEXIT_FAILURE);
        }
/* -main_library_call_end */

/* -main_after_lib_call */
/*
      ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      + Copy the entire -main_after_lib_call section as is.
      ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/
/* Shove the database data down the output pipe */
      kstrcpy(mpiData->inDataFileName,mpiData->outDataFileName);
           /* out here is in for next */
      kstrcpy(mpiData->outDataFileName,"");
      mpiData->currentProcessNum++;
      outDataFile = kfopen(clui_info->outData_file,"w");
      ret = kfwrite(mpiData, sizeof(mpiStruct), 1, outDataFile);
      if (ret != 1)
      {
       kerror(lib, rtn, "Unable to write MPI data to output stream");
       kexit(KEXIT_FAILURE);
      }

/* close up files and free memory */
      kfree(parJobBinDir);
      kfree(abstractCMD);
      kfree(mpiData);
      kfclose(outDataFile);
/* -main_after_lib_call_end */


      kexit(KEXIT_SUCCESS);
}


/*-------------------------------------------------------
|
| Routine Name: YourProgram_usage_additions
|
|     Purpose: Prints usage additions in YourProgram_usage routine
|
|       Input: None
|
|      Output: None
|   Written By: ghostwriter -oname YourProgram
|        Date: Jun 26, 1997
| Modifications:
```

```
|
------------------------------------------------------------*/
void YourProgram_usage_additions(void)
{
     kfprintf(kstderr, "Appropriate message for YourProgram.");

/* -usage_additions */
/* -usage_additions_end */

}
/*------------------------------------------------------------
|
| Routine Name: YourProgram_free_args
|
|     Purpose: Frees CLUI struct allocated in YourProgram_get_args()
|
|       Input: None
|
|      Output: None
|   Written By: ghostwriter -oname YourProgram
|         Date: Jun 26, 1997
| Modifications:
|
------------------------------------------------------------*/
/* ARGSUSED */
void
YourProgram_free_args(
  kexit_status status,
  kaddr       client_data)
{

     /* do the wild and free thing */
     if (clui_info != NULL)
     {
          kfree_and_NULL(clui_info->inData_file);
          kfree_and_NULL(clui_info->outData_file);
          kfree_and_NULL(clui_info);
     }

/* -free_handler_additions */
/* -free_handler_additions_end */
}
```

C.2. Template for the Pkroutine

```
#include "YourPkroutine.h"

/*-------------------------------------------------------
|
| Routine Name: YourPkroutine
|
|     Purpose:
|
|     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|     + This routine is the top-level parallel executable.
|     + For purposes of compatability with future versions of
|     + Khoros, our suggestion is that this routine do
|     + little more than parse arguments, open the input
|     + object(s), and pass them to the lpkroutine.
|     + Khoros' future plan for parallel executions from
|     + Cantata is that the kroutine itself will be
|     + executable on the remote computer.
|     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|
|       Input: Input file name
|              Output file name
|
|      Output:
|     Returns:
|
|   Written By:
|        Date: May 25, 1997
| Modifications:
|
-------------------------------------------------------*/

void main(int argc, char **argv)
{

 /* Variables for kerror */
 char *lib = "NONE";
 char *rtn = "YourPkroutine";

 /* Khoros variables */
 kobject src_obj, dst_obj;

 kdist_init(&argc, &argv);
 rank = kdist_rank();
 size = kdist_size();

 printf("%d : Processing YourPkroutine from %d processors \n", rank, size);
```

```
 /* Check for correct # of command line arguments */
 if(argc != NARGS && rank == ROOT)
  {
   kerror(lib, rtn, "\nChecking args: argc = %d \n",argc);
   printUsage();
   kexit(KEXIT_FAILURE);
  }

/*
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     + Following is an example of parsing the command line
     + for the parallel fft.
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/
 /* Process command line arguments                                  */
 /* Minimum required - input file and output file                   */
 /*              - fft_flag == forward or inverse                   */
 /*              - trans_flag == output transposed or original order */
 for(i = 1; i < argc; i++)
  {
   switch(argv[i][1])
    {
    case 'i':
     sprintf(inFileName,"%s",argv[++i]);
     break;
    case 'o':
     sprintf(outFileName,"%s",argv[++i]);
     break;
    case 'f':
     fft_flag = atoi(argv[++i]);
     if (fft_flag != FORWARD && fft_flag != INVERSE)
      {
       if (rank == ROOT)
        {
         kerror(lib, rtn,
              "\nChecking fft_flag: flag = %d \n", fft_flag);
         printUsage();
        }
       kexit(KEXIT_FAILURE);
      }
     break;
    case 't':
     trans_flag = atoi(argv[++i]);
     if (trans_flag != 0 && trans_flag != 1)
      {
       if (rank == ROOT)
        {
         kerror(lib, rtn,
              "\nChecking trans_flag: flag = %d \n", trans_flag);
         printUsage();
```

```
            }
           kexit(KEXIT_FAILURE);
          }
         break;
        default:
         if (rank == ROOT)
          {
           kerror(lib, rtn, "\nChecking input flags: incorrect\n");
           printUsage();
          }
         kexit(KEXIT_FAILURE);
        }
    }

/*
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     + Here are the Khoros distributed data services calls
     + to open a KDF object whose data may later be
     + distributed. Normal kpds calls may also be used here
     + for objects which do not require distribution.
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/

 /* Create distributed kobjects */
 if ((src_obj = kpds_open_distributed_input_object(inFileName))
          == KOBJECT_INVALID)
   {
    kerror(lib, rtn, "Unable to open input object: %s\n", inFileName);
    kexit(KEXIT_FAILURE);
   }

 if ((dst_obj = kpds_open_distributed_output_object(outFileName))
          == KOBJECT_INVALID)
   {
    kerror(lib, rtn, "Unable to open output object: %s\n", outFileName);
    kexit(KEXIT_FAILURE);
   }

/*
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     + The call to your routine for distributing/processing
     + the data object.
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/
 i_error = Yourlpkroutine(src_obj, dst_obj, ... other args as required ...);
 if (i_error)
   {
    kerror(lib, rtn, "\n***Failure in Yourlpkroutine %d***\n");
    kexit(KEXIT_FAILURE);
   }
```

```
/*
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     + Close the data objects.
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/
  kpds_close_distributed_object(src_obj);
  kpds_close_distributed_object(dst_obj);

  kexit(KEXIT_SUCCESS);

} /* end main */
```

C.3. Lpkroutine and Lroutine

The lpkroutine's function is to distribute data if necessary and pass the raw data to the lroutine. This may be accomplished via calls to Khoros distributed data services, in which case the KDF object structure will be available on all processors. Alternatively, the value segment data can be read in and distributed by MPI calls. There are no templates for these levels as code will vary depending on the needs and implementation choices of the developer. The pkcfft object in Para_tools is an example of one possible way to divide responsibilities.

Integration of already existing parallel code into the Khoros environment can follow much the same format described here. A call to the existing code would be made from the lroutine level, passing data previously distributed and extracted from the KDF object.

9. Critical Design Review For Advanced Signal Processing, Integration of Data Compression into RLSTAP

Marc Friedman
Maui High Performance Computing Center (MHPCC)
Joe Fogler
Brendan Bradley
Albuquerque High Performance Computing Center (AHPCC)

13 May, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

1 Introduction

The Advanced Signal Processing (ASP) program is a DARPA sponsored activity for studying advanced processing techniques and technologies for next generation air early warning (AEW) platforms.[1] A key technology area for this activity is software tools and methodologies for collaborative algorithm development.

Khoral Research Incorporated (KRI), a spinoff company from the University of New Mexico Electrical and Computer Engineering Department, has created a software integration and development environment for information processing, data exploration and visualization called Khoros.[2] Khoros is a comprehensive software system with a rich set of tools usable by both end-users and application developers. Included in these tools is a graphical programming application called Cantata which gives users the ability to construct complex algorithms by interconnecting iconic representations, called glyphs, of processing functions in a terminal window called a workspace, using mouse point-and-click operations. Khoros has become a de-facto standard for collaborative algorithm development in the Department of Defense automatic target recognition (ATR) community.

The Rome Laboratory Space-Time Adaptive Processing (RLSTAP) tool, utilizing Khoros and its graphical programming environment Cantata, represents a state-of-the-art development environment for clutter modeling and radar simulation for advanced early warning (AEW) applications, and has found use by researchers working on Navy E-2C and the Air Force E-3A upgrades. Written initially in Khoros version 1.0, RLSTAP is currenly being ported to the latest Khoros release, version 2.1, in a separate development.

Signal processing algorithms required by AEW applications are computationally intensive and utilize large data sets for experimentation and validation. These are driving a need for distributed parallel processing resources such as those available at the Maui High Performance Computing Center (MHPCC).

With large data sets for AEW applications archived at the MHPCC in Kihei, Hawaii, and researchers located at diverse geographical locations throughout the country, there is a frequent need to distribute data from the archive in Maui to distant researchers. The primary modality for this data movement is the ethernet. The combination of large data sets and limited communication bandwidth drive a need for data compression for such file transfers. Lossless compression is sought to ensure that algorithm performance validation can be performed accurately. Two types of data are in need of this compression – Radar Surveillance Technology Experimental Radar) RSTER-like radar signals and synthetic aperture radar (SAR) data.

The purpose of this effort is to investigate lossless data compression strategies for RSTER-like and SAR data, and develop a set of software applications for integration into the new RLSTAP (compatible with Khoros 2.1) environment that perform efficient lossless compression and uncompression. Versions of these applications will also be generated that operate standalone without the Khoros environment.

2 Scope

2.1 System Overview

The vast majority of programs written for or supplied with the Khoros environment are *kroutines*. These include all data processing programs such as image processing and signal processing routines. Kroutines are fully usable from within the Cantata graphical programming environment, but generally do not display any graphics or images; they simply input data, process it, and output results.

[1] G. W. Titi, An Overview of the ARPA/NAVY Mountaintop Program, Proceedings IEEE Adaptive Antenna Systems Symposium, (1994).

[2] See KRI's website at *http://www.khoral.com/* for more details.

Kroutines may also be invoked without Cantata as ordinary applications under the Unix operating system, if desired. By convention, kroutines usually have names that begin with the letter *k* when invoked from a Unix command line, although their names may appear slightly different when represented graphically (in an iconic form called a glyph) in a Cantata workspace.

It is convenient to partition the data compression routines into two categories – (1) routines that apply to RSTER-like data, and (2) routines that apply to SAR data. This partitioning is opted since it cannot be assumed that the data formats of RSTER-like and SAR data will be similar, nor can it be assumed that the same data compression technique will work for both types.

2.1.1 Software routines for RSTER-like data

Files in the database of RSTER-like data are stored as Matlab version 4.2 matrix files.[3]

A high percentage of the space required to store RSTER-like data files is taken up by a set of complex arrays given names of the form *cpi#* (Coherent Pulse Interval) where # is a cardinal number. These are the items in the files upon which data compression will be focused. In subsequent discussions, RSTER-like data files will be referred to simply as *RSTER* files, although it should be noted that these files also contain small amounts of ancillary data in the form of arrays and scalars.

Routine name	Description
rster_compress	Standalone executable for compressing RSTER file
rster_uncompress	Standalone executable for uncompressing RSTER file
RSTERcompress	Khoros 2.1 executable for compressing RSTER file
RSTERuncompress	Khoros 2.1 executable for uncompressing RSTER file

Each of the above routines acts as a driver for an associated library routine where the actual compression and decompression work is performed.

2.1.2 Software routines for SAR data

SAR data is available from a number of sources in a variety of formats. However, for data distribution purposes, it can be categorized into two domains: phase histories and formed images.

Phase histories represent SAR data in raw form which is typically stored on magnetic tape along with platform flight data such as velocity, altitude, and sensor-to-ground geometry such as depression angle and squint angle. Each phase history datum is complex-valued consisting of a real (I) and imaginary (Q) component. Phase history data collected over a synthetic aperture represent range and range-rate radar returns in a two-dimensional matrix of complex values obtained over a large rectangular patch on the ground. The precision with which collected phase history data is initially stored is often 16 bits or less for each real component and a similar number of bits for its imaginary counterpart.

Image formation is the process by which phase histories are converted via FFTs and other processes to so-called formed images. These are complex valued, but their magnitudes are displayable and interpretable by the human eye. A great deal of effort must be spent in the image formation process to ensure that the resulting data is properly focussed, correctly calibrated, and geometrically accurate. For these and other reasons, most users find the formed image domain of greater use than phase histories.

SAR data compatible with the RLSTAP environment will not be made available for this devleopment. Therefore, unclassified SAR data from other sources will be sought, including rural clutter scenes from the Advanced Detection Technology Sensor (ADTS), for data compression algorithm development. However, there is no assurance that this data will be representative of actual SAR data that may come into use beyond the timeframe and scope of this task. Given these assumptions,

[3]The file format for Matlab matrix files is described in the External Interface Guide supplied with Matlab software distributions.

the routines listed in the following table will be developed. For convenience the formed image SAR data files to be utilized in this development are referred to simply as *SAR* files.

routine name	Description
sar_compress	Standalone executable for compressing SAR file
sar_uncompress	Standalone executable for uncompressing SAR file
SARcompress	Khoros 2.1 executable for compressing SAR file
SARuncompress	Khoros 2.1 executable for uncompressing SAR file

2.2 Limitations

The data compression algorithms applied to this problem will be tuned to the statistics of data identified for this task, and may not be broadly applicable to other types of data. Furthermore, although the algorithms will be efficient and produce a substantial amount of compression, they may not be optimal if, for example, better algorithms are available from other sources, but are eliminated from consideration for proprietary reasons.

The RSTER data compress and uncompress routines will be developed using data intended for RLSTAP and will be fully compatible with the RLSTAP Khoros 2.1 environment. However, the SAR data compress and uncompress routines will be developed utilizing SAR data that is not intended for the RLSTAP environment. There is a reasonable expectation that useful amounts of compression will be achievable using other types of SAR data, but this cannot be assured and will not be tested. Also, the file format of the non-RLSTAP SAR data to be used for development may be different than that of the SAR data to be eventually used with RLSTAP. This means that some modification to the low-level I/O routines used by these routines may be required in a future development beyond the scope of the current task in order to achieve full compatibility with RLSTAP.

Since RLSTAP is currently supported under two operating environments, AIX on IBM RS6000 computers, and SunOS/Solaris on SUN SPARC computers, the file import, export, and data compression routines will be designed for compatibility with the these platforms. The routines provided may be compatible with additional platforms, but this will not be tested.

3 Reference Documents

Software routines to be written for use in Khoros will be developed using case tools built into Khoros. These include *craftsman* which is used for the creation and management of collections of routines called toolboxes, *composer* which is used to edit, manipulate, and compile existing software objects (e.g. kroutines), and *guise* which is used to edit graphical user interface (GUI) objects (e.g. panes and forms). These case tools are described in Chapters 2, 3, and 4, respectively, in the *Khoros 2.1 Toolbox Programming Guide.*

All file I/O performed by routines written in Khoros will be implemented using Khoros data services routines. These are described in the *Khoros 2.1 Data Services Guide.*

Documentation of the data compression tools will be provided in three forms – man pages, on-line help, and a printed manual.

Man pages will serve as on-line documentation for both users and programmers, describing functionality and usage of the various programs and utilities. Khoros man pages are accessable to the user via the *kman* command which is similar in operation to the standard Unix *man* command.

On-line help is accessible by the user via a *Help* button on the graphical user interface pane that can be displayed from within the Cantata graphical programming environment. The information provided through the *Help* button is similar to that obtainable from man pages.

The printed manual will provide a hardcopy representation of the documentation and encapsulate much of the documentation into an integrated whole. The tools to support printed manual

documentation are embedded within the Khoros *imake* system, which provides several macros for including man pages, code segments, and function descriptions.

A description of documentation facilities within Khoros is provided in Chapter 6 of the *Khoros 2.1 Toolbox Programming Manual*. Examples of documentation generated using these built-in facilities can be found throughout that manual.

4 Design

4.1 Software Development Plan

Software development will be performed in two stages. In the first stage, software tools will be developed for compressing, and uncompressing RSTER radar data. In the second stage, software tools will be developed for compressing, and uncompressing SAR data.

The development of compression and uncompression routines for RSTER data will proceed in the following steps: (1) data analysis, (2) algorithm experimentation and design, (3) baseline algorithm selection, (4) coding of library routines for compression and uncompression, (5) coding of standalone driver programs that call the library routines, (6) informal testing of the software for internal purposes, (7) coding of Khoros-compatible versions of the driver programs, and (8) generation of man pages, on-line help, and printed manual documentation.

The above steps will be repeated in the second stage of development for the compression and uncompression of SAR data.

4.2 Data Description

4.2.1 RSTER Matrix Files

Each RSTER matrix file is composed of one or more *cpi* matrices, and a number of ancillary arrays and scalar values, all stored as a sequence of Matlab version 4.2 matrices. A Matlab version 4.2 matrix file contains one or more matrices each consisting of a 20-byte header, followed by a matrix name string, followed by actual matrix data. Each matrix is stored sequentially and contiguously in the file.

The Matlab header consists of five 4-byte signed integers that define, in order, (1) the matrix type, (2) number of rows in the matrix, (3) number of columns in the matrix, (4) whether the matrix is real or complex, and (5) the length of the matrix name including a NULL terminator character. The matrix type encodes the precision and data type of the matrix data, the machine architecture upon which the data was generated, and whether the matrix is sparse, numeric, or contains text. All the matrices in this application are either numeric or textual.

All matrices are stored as type double (real or complex) data values in memory within Matlab. However, to reduce storage requirements when large matrices are stored to files using the Matlab *save* command, data is converted to a data type that requires fewer bytes-per-item where possible according to an internal algorithm. For example, if all data within a real type double matrix are integral values (i.e. representable as integers), and are bounded by the representable range of 32-bit signed integers, the data is converted to and stored as 32-bit signed integers automatically when saved. If all data are integral and bounded by the representable range of 16-bit signed integers, the data is converted to and stored as 16-bit signed integers. Complex data, are similarly converted where the real and imaginary components are tested independently against the bounds.

The *cpi* arrays, which constitute a high percentage of the storage requirements of a RSTER file, are complex data with real and imaginary components each stored as signed integers. The Matlab format places all the real components of a matrix first in the file followed by all its imaginary components.

Each RSTER file contains at least one *cpi#* array where the first such array is given the name *cpi1*. Other matrices are present in the file that represent ancillary information. Many of these ancillary matrices are optional and therefore may not be present in every RSTER file. The ancillary matrices included in the following table are merely representative of a typical RSTER file.

Matrix name	Shape	Type	Integral Values
muxtype	scalar	real	yes
trecord	scalar	real	yes
wrecord	scalar	real	yes
cpitype	vector	real	yes
cpiidx	vector	real	yes
scanidx	vector	real	yes
npulses	vector	real	yes
fxmit	vector	real	no
pri	vector	real	no
tpulse	vector	real	no
elxmit	vector	real	no
azxmit	vector	real	no
cpitime	matrix	real	no
cpi1	matrix	cplx	yes
cpi2	matrix	cplx	yes
cpi3	matrix	cplx	yes
cpi4	matrix	cplx	yes
cpi5	matrix	cplx	yes
flgRound	scalar	real	yes
flgChopTrans	scalar	real	yes
calflag	scalar	real	yes
flgEqualize	scalar	real	yes
flgCalibrate	scalar	real	yes
nCPI	scalar	real	yes
CurrFreq	scalar	real	no
ant_tilt	scalar	real	no
rster_alt	scalar	real	no
rster_lat	scalar	real	no
rster_lon	scalar	real	no
rster_pow	scalar	real	no
info	matrix	text	n/a

The number of cpi arrays in the file is stored in the scalar variable *nCPI*. The vectors *cpitype, cpiidx, scanidx, npulses, fxmit, pri, tpulse, elxmit, and azxmit* are all of length *nCPI*. The matrix *cpitime* is composed of *nCPI* rows and six columns and contains time information stored as real numeric values. The *info* array is textual and is composed of a number of rows of 80-column text strings with each character represented as an ASCII code stored one per each matrix item.

4.2.2 Compressed RSTER Files

The format for compressed RSTER files will depend on the nature of the chosen data compression algorithm, and may be opaque to the user. That is, the algorithm will encode the file to the extent that the file contents may not be interpretable by any means other than to uncompress the file back to its original form, a RSTER matrix file, using the routines *rster_uncompress* or its Khoros-compatible equivalent *RSTERuncompress*.

4.2.3 SAR Data Files

Data will be sought in the form of rural SAR scenes from the ADTS sensor collection. The formed image domain v-v polarized channel data will be extracted and converted to a Matlab 4.2-compatible file format. This will permit reuse of the low-level I/O routines developed for the RSTER compress and uncompress routines. Again, this data is for demonstration purposes only, and will not be RLSTAP compatible. Additional I/O routines may be required beyond the timeframe and scope of this task to support other SAR data formats.

4.2.4 SAR Compressed Files

The format for compressed SAR files will depend on the nature of the chosen data compression algorithms, and may be opaque to the user. That is, the algorithm will encode the file to the extent that the file contents may not be interpretable by any means other than to uncompress the file back to its original form, a SAR matrix file, using the routines *sar_uncompress* or its Khoros-compatible equivalent *SARuncompress*.

4.3 Modules

The software modules for compressing, and uncompressing data files are described in the following sections. The standalone routines are described first, followed by the Khoros-compatible versions of the same.

4.3.1 rster_compress

Purpose:

The purpose of the *rster_compress* standalone application is to input a RSTER Matlab file, compress the data in the file, and output the resulting RSTER file in compressed form.

Process:

rster_compress is a driver program that opens the input RSTER Matlab file for reading and the output file for writing using the C-language system routines *fopen* and *fclose*.

The input and output file descriptors are then passed to the *lrster_compress* library routine where the actual read-compress-write operations take place.

The data compression algorithm is executed on each cpi array and the results are written to the output object. Selected attributes from the input file are also copied to the output file, and the procedure completes.

Details of the compressed RSTER file format are dependent upon the data compression algorithm used. The routines *rster_uncompress* and its Khoros-compatible equivalent *RSTERuncompress* will be capable of reading and uncompressing this file format to produce standard RSTER Matlab files.

Interfaces:

The *rster_compress* routine will utilize command line interface code that mimics a subset of the command line user interface (CLUI) facilities built into Khoros. However, only those command line arguments and features described here will be supported.

Command line arguments specific to the *rster_compress* routine are as follows:

Argument	Type	Description
-i	str	input RSTER matrix file
-o	str	output compressed RSTER file
-U	boolean	request usage (Khoros style)

Variable definitions:

The *rster_compress* routine utilizes the variables *src*, and *dest*.

The variable *src*, of type *FILE **, is a pointer to a structure returned by a call to *fopen*, and is used to access the input file.

The variable *dest*, of type *FILE **, is a pointer to a structure returned by a call to *fopen*, and is used to access the output file.

Routines called by rster_compress:

The *lrster_compress* library routine is opaque from command line execution of the *rster_compress* routine. Its formal parameters are as follows:

Argument	Type	Description
src	FILE *	Input file descriptor
dest	FILE *	Output file descriptor

Other library routines are called from *lrster_compress* and are described in later subsections. The names of these routines are

```
fget_next_matrix
fput_next_matrix
free_matrix
```

4.3.2 rster_uncompress

Purpose:

The purpose of the *rster_uncompress* standalone application is to input a compressed RSTER file, uncompress the data in the file, and output the result as a standard RSTER Matlab file.

Process:

rster_uncompress is a driver program that opens the compressed input file for reading and the output RSTER Matlab file for writing using the C-language system routines *fopen* and *fclose*.

The input and output file descriptors are then passed to the *lrster_uncompress* library routine where the actual read-uncompress-write operations take place.

The data uncompression algorithm is executed on each cpi array and the results are written to the output object. Selected attributes from the input file are also copied to the output file, and the procedure completes. The resulting file is a standard RSTER Matlab file.

Interfaces:

The *rster_uncompress* routine will utilize command line interface code that mimics a subset of the command line user interface (CLUI) facilities built into Khoros. However, only those command line arguments and features described here will be supported.

Command line arguments specific to the *rster_uncompress* routine are as follows:

Argument	Type	Description
-i	str	input compressed RSTER file
-o	str	output RSTER matrix file
-U	boolean	request usage (Khoros style)

Variable definitions:

The *rster_uncompress* routine utilizes the variables *src*, and *dest*.

The variable *src*, of type *FILE **, is a pointer to a structure returned by a call to *fopen*, and is used to access the input file.

The variable *dest*, of type *FILE **, is a pointer to a structure returned by a call to *fopen*, and is used to access the output file.

Routines called by rster_uncompress:

The *lrster_uncompress* libary routine is opaque from command line execution of the *rster_uncompress* routine. Its formal parameters are as follows:

Argument	Type	Description
src	FILE *	Input file descriptor
dest	FILE *	Output file descriptor

Other library routines are called from *lrster_uncompress* and are described in later subsections. The names of these routines are

```
fget_next_matrix
fput_next_matrix
free_matrix
```

4.3.3 fget_next_matrix

Purpose:

The library routine *fget_next_matrix* mimics the functionality of the Matlab script *matGetNextMatrix*. That is, it reads the matrix at the current file position in an opened Matlab file into memory.

Process:

The routine *fget_next_matrix* uses the standard C-language file I/O routine *fread* to read in the header, matrix name, and data from a matrix into memory, and returns a pointer to the matrix structure.

Interfaces and variable definitions:

Formal parameters for the *fget_next_matrix* routine are given in the following table:

Argument	Type	Description
fd	FILE *	Input file descriptor

The routine *fget_next_matrix* returns a pointer to a structure of type *matrix* which is defined as follows:

```
typedef struct _matrix {
  char name[20];  /* matrix name */
  int mmach;      /* machine type */
  int dtype;      /* data type */
  int mtype;      /* matrix type */
  int m;          /* row dimension */
  int n;          /* col dimension */
  void *pr;       /* pointer to real part */
  void *pi;       /* pointer to imag part */
```

Machine type, data type, and matrix type *defines* associated with the *matrix* structure are:

```
/* data types */
#define DTYPE_UNKNOWN -1
#define DTYPE_DOUBLE  0
#define DTYPE_FLOAT   1
#define DTYPE_INT     2
#define DTYPE_SHORT   3
#define DTYPE_USHORT  4
#define DTYPE_UBYTE   5

/* special type used for compression */
#define DTYPE_INT3    6
```

```
/* matrix types */
#define MTYPE_UNKNOWN -1
#define MTYPE_REAL    0
#define MTYPE_TEXT    1
#define MTYPE_SPARSE  2
#define MTYPE_COMPLEX 3

/* machine types */
#define MMACH_UNKNOWN       -1
#define MMACH_LITTLE_ENDIAN  0
#define MMACH_BIG_ENDIAN     1
```

Other routines called:

`fread`

4.3.4 fput_next_matrix

Purpose:

The library routine *fput_next_matrix* mimics the functionality of the Matlab script *matPutMatrix*. That is, it writes a matrix from memory beginning at the current file position in an opened Matlab-format file.

Process:

The routine *fput_next_matrix* uses the standard C-language file I/O routine *fwrite* to write out the header, matrix name, and data from a matrix in memory.

Interfaces and variable definitions:

Formal parameters for the *fput_next_matrix* routine are given in the following table:

Argument	Type	Description
mat	matrix *	Pointer to matrix struct
fd	FILE *	Output file descriptor

The routine *fput_next_matrix* returns a type integer status value which is TRUE (1) if successful, or FALSE (0) if an error occurred.

Other routines called:

`fwrite`

4.3.5 free_matrix

Purpose:

The library routine *free_matrix* deallocates the memory associated with the data buffers and structure of a matrix.

Process:

The routine *free_matrix* frees the memory associated with the data buffers and matrix structure. If any of these pointers are NULL, no action is taken to free that associated pointer.

Interfaces and variable definitions:

Formal parameters for the *free_matrix* routine are given in the following table:

Argument	Type	Description
mat	matrix *	Pointer to matrix struct

The routine *free_matrix* returns void.

Other routines called:

`free`

4.3.6 RSTERcompress

Purpose:

The purpose of the Khoros kroutine *RSTERcompress* is to input a RSTER Matlab file, compress the data in the file, and output a compressed RSTER file.

Process:

RSTERcompress is a driver program that opens the input Matlab file for reading and the output file for writing.

The input and output file descriptors are then passed to the *lRSTERcompress* library routine where the actual read-compress-write operations take place.

The data compression algorithm is executed on each cpi array and the results are written to the output object. Selected attributes from the input file are also copied to the output file, and the procedure completes.

Details of the compressed RSTER file format are dependent upon the data compression algorithm used. All that can be specified at this time is that the routine *RSTERuncompress* will be capable of reading the file and converting it back to an uncompressed RSTER Matlab file.

Interfaces:

The *RSTERcompress* kroutine utilizes the command line user interface (CLUI) facilities built into Khoros and detailed in the *Khoros 2.1 Toolbox Programming Guide.*

Command line arguments specific to the *RSTERcompress* routine are as follows:

Argument	Type	Description
-i	infile	input data object
-o	outfile	Compressed output file object

Variable definitions:

The *rster_compress* routine utilizes the variables *src*, and *dest.*

The variable *src*, of type *kobject*, is a pointer to a structure returned by a call to *kpds_open_input_object*, and is used by Khoros data services routines to access the input object.

The variable *dest*, of type *kobject*, is a pointer to a structure returned by a call to *kpds_open_output_object*, and is used by Khoros data services routines to access the output object.

Routines called by RSTERcompress:

The *lRSTERcompress* libary routine is opaque from command line execution of the *RSTERcompress* kroutine. Its formal parameters are as follows:

Argument	Type	Description
src	kobject	Input object descriptor
dest	kobject	Output object descriptor

Other library routines are called from *lRSTERcompress* and are described in later subsections. The names of these routines are

`lget_next_matrix`
`lput_next_matrix`
`lfree_matrix`

4.3.7 RSTERuncompress

Purpose:

The purpose of *RSTERuncompress* is to input a compressed RSTER file, uncompress the data in the file, and output a Matlab RSTER file.

Process:

RSTERuncompress is a driver program that opens the compressed input file for reading and the output RSTER file for writing using the Khoros data services routines *kpds_open_input_object* and *kpds_open_output_object*, respectively.

The input and output file descriptors are then passed to the *lRSTERuncompress* library routine where the actual read-uncompress-write operations take place.

The data uncompression algorithm is executed on each cpi array and the results are written to the output object. Selected attributes from the input file are also copied to the output file, and the procedure completes.

Details of the compressed RSTER file format are dependent upon the actual data compression algorithm used. However, regardless of the compression algorithm utilized, the kroutine *RSTERuncompress* and the standalone version *rster_uncompress* will be capable of reading compressed files generated by the kroutine *RSTERcompress*.

Interfaces:

The *RSTERuncompress* routine utilizes the command line user interface (CLUI) facilities built into Khoros and detailed in the *Khoros 2.1 Toolbox Programming Guide*. Included in this interface is the automatic generation of usage activated by the *-U* command line option. Since this is automatically generated, it is implied and therefore not listed among the command line arguments.

Command line arguments specific to the *RSTERuncompress* routine are as follows:

Argument	Type	Description
-i	infile	Compressed input data object
-o	outfile	output file object

Variable definitions:

The *RSTERuncompress* kroutine utilizes the variables *src*, and *dest*.

The variable *src*, of type *kobject*, is a pointer to a structure returned by a call to *kpds_open_input_object*, and is used by Khoros data services routines to access the input object.

The variable *dest*, of type *kobject*, is a pointer to a structure returned by a call to *kpds_open_output_object*, and is used by Khoros data services routines to access the output object.

Routines called by RSTERuncompress:

The *lRSTERuncompress* libary routine is opaque from command line execution of the *RSTERuncompress* routine. Its formal parameters are as follows:

Argument	Type	Description
src	kobject	Input object descriptor
dest	kobject	Output object descriptor

Other library routines are called from *lrster_uncompress* and are described in later subsections. The names of these routines are

```
lget_next_matrix
lput_next_matrix
lfree_matrix
```

4.3.8 lget_next_matrix

Purpose:

The library routine *lget_next_matrix* mimics the functionality of the Matlab script *matGetNextMatrix*. That is, it reads the matrix at the current file position in an opened Matlab file into memory. It is virtually identical to the standalone library routine *fget_next_matrix* but uses the Khoros versions of system library routines.

Process:

The routine *lget_next_matrix* uses the Khoros file I/O routine *kfread* to read in the header, matrix name, and data from a matrix into memory, and returns a pointer to the matrix structure.

Interfaces and variable definitions:

Formal parameters for the *lget_next_matrix* routine are given in the following table:

Argument	Type	Description
fd	kfile *	Input file descriptor

The routine *lget_next_matrix* returns a pointer to a structure of type *matrix* which was defined previously in the subsection detailing the routine *fget_next_matrix*.

Other routines called:

`kfread`

4.3.9 lput_next_matrix

Purpose:

The library routine *lput_next_matrix* mimics the functionality of the Matlab script *matPutMatrix*. That is, it writes a matrix from memory beginning at the current file position in an opened Matlab-format file. It is virtually identical in function to the standalone library routine *fput_next_matrix* but utilizes Khoros versions of system I/O library routines.

Process:

The routine *lput_next_matrix* uses the Khoros file I/O routine *kfwrite* to write out the header, matrix name, and data from a matrix in memory.

Interfaces and variable definitions:

Formal parameters for the *lput_next_matrix* routine are given in the following table:

Argument	Type	Description
mat	matrix *	Pointer to matrix struct
fd	kfile *	Output file descriptor

The routine *lput_next_matrix* returns a type integer status value which is TRUE (1) if successful, or FALSE (0) if an error occurred.

Other routines called:

`kfwrite`

4.3.10 lfree_matrix

Purpose:

The library routine *lfree_matrix* deallocates the memory associated with the data buffers and structure of a matrix. It is virtually identical to the standalone version *free_matrix* except that the Khoros version of *free*, (*kfree*) is utilized to deallocate memory.

Process:

The routine *lfree_matrix* frees the memory associated with the data buffers and matrix structure. If any of these pointers are NULL, no action is taken to free that associated pointer.

Interfaces and variable definitions:

Formal parameters for the *lfree_matrix* routine are given in the following table:

Argument	Type	Description
mat	matrix *	Pointer to matrix struct

The routine *lfree_matrix* returns void.

Other routines called:

`kfree`

4.3.11 sar_compress

Purpose:

The purpose of the *sar_compress* standalone application is to input a SAR file, compress the data in the file, and output the resulting SAR file in compressed form. The file format of the input SAR file will be Matlab 4.2. This will permit the reuse of file i/o routines developed for reading RSTER files. The I/O aspects of *sar_compress* may need to be modified at a later time to accomodate the actual SAR data to be eventually used in the RLSTAP environment. However, such modifications are beyond the scope of the current task.

Two compression algorithms will be supported, *arithmetic coding* and *huffman coding*. A command line option will be utilized to select the desired algorithm.

Process:

sar_compress is a driver program that opens the input SAR Matlab file for reading and the output file for writing using the C-language system routines *fopen* and *fclose*.

The input and output file descriptors and algorithm option flag are then passed to the *lsar_compress* library routine where the actual read-compress-write operations take place.

The data compression algorithm is executed on the sar data array and the results are written to the output object. Selected attributes from the input file are also copied to the output file, and the procedure completes.

Details of the compressed SAR file format are dependent upon the data compression algorithm used. The routines *sar_uncompress* and its Khoros-compatible equivalent *SARuncompress* will be capable of reading and uncompressing this file format to produce a Matlab 4.2 compatible file.

Interfaces:

The *sar_compress* routine will utilize command line interface code that mimics a subset of the command line user interface (CLUI) facilities built into Khoros. However, only those command line arguments and features described here will be supported.

Command line arguments specific to the *rster_compress* routine are as follows:

Argument	Type	Description
-i	str	input SAR matrix file
-t	int	(1) Huffman, (2) Arithmetic
-o	str	output compressed SAR file
-U	boolean	request usage (Khoros style)

Variable definitions:

The *sar_compress* routine utilizes the variables *src*, *dest*, and *atype*.

The variable *src*, of type *FILE **, is a pointer to a structure returned by a call to *fopen*, and is used to access the input file.

The variable *dest*, of type *FILE **, is a pointer to a structure returned by a call to *fopen*, and is used to access the output file.

The variable *atype*, of type *integer*, determines which algorithm type to utilize. If *atype* is 1, Huffman coding is to be used. If *atype* is 2, Arithmetic coding is to be used.

Routines called by sar_compress:

The *lsar_compress* library routine is opaque from command line execution of the *sar_compress* routine. Its formal parameters are as follows:

Argument	Type	Description
src	FILE *	Input file descriptor
atype	integer	(1) Huffman, (2) Arithmetic
dest	FILE *	Output file descriptor

A number of library routines are called from *lsar_compress* and are described in other subsections. The names of these routines are

```
huffman_encode
arith_encode
fget_next_matrix
fput_next_matrix
free_matrix
```

4.3.12 sar_uncompress

Purpose:

The purpose of the *sar_uncompress* standalone application is to input a compressed SAR file, uncompress the data in the file, and output the result as a Matlab-compatible SAR file.

Process:

sar_uncompress is a driver program that opens the compressed input file for reading and then outputs a SAR Matlab file for writing using the C-language system routines *fopen* and *fclose*.

The input and output file descriptors are then passed to the *lrster_uncompress* library routine where the actual read-uncompress-write operations take place.

The data uncompression algorithm is executed on the sar data array and the results are written to the output object. Selected attributes from the input file are also copied to the output file, and the procedure completes. The resulting file is a Matlab 4.2 compatible SAR file.

Interfaces:

The *sar_uncompress* routine will utilize command line interface code that mimics a subset of the command line user interface (CLUI) facilities built into Khoros. However, only those command line arguments and features described here will be supported.

Command line arguments specific to the *sar_uncompress* routine are as follows:

Argument	Type	Description
-i	str	input compressed SAR file
-o	str	output SAR matrix file
-U	boolean	report usage (Khoros style)

Variable definitions:

The *sar_uncompress* routine utilizes the variables *src*, and *dest*.

The variable *src*, of type *FILE **, is a pointer to a structure returned by a call to *fopen*, and is used to access the input file.

The variable *dest*, of type *FILE **, is a pointer to a structure returned by a call to *fopen*, and is used to access the output file.

Routines called by sar_uncompress:

The *lsar_uncompress* libary routine is opaque from command line execution of the *sar_uncompress* routine. Its formal parameters are as follows:

Argument	Type	Description
src	FILE *	Input file descriptor
dest	FILE *	Output file descriptor

A number of library routines are called from *lsar_uncompress* and are described in other subsections. The names of these routines are

```
huffman_decode
arith_decode
fget_next_matrix
fput_next_matrix
free_matrix
```

4.3.13 huffman_encode

Purpose:

The library routine *huffman_encode* performs data compression using a Huffman coding algorithm.

Process:

The routine *huffman_encode* accesses the data using pointers in the memory matrix structs that are passed to the routine. The memory matrix struct type was described in a previous section.

Interfaces and variable definitions:

Formal parameters for the *huffman_encode* routine are given in the following table:

Argument	Type	Description
imat	matrix *	Pointer to input matrix struct
omat	matrix *	Pointer to output matrix struct

The routine *huffman_encode* returns a type integer status value which is TRUE (1) if successful, or FALSE (0) if an error occurred.

Other routines called:

none

4.3.14 huffman_decode

Purpose:

The library routine *huffman_decode* decompresses data that was previously compressed using the Huffman coding algorithm utilized in the routine *huffman_encode*.

Process:

The routine *huffman_decode* accesses the data using pointers elements of the memory matrix structs that are passed to the routine.

Interfaces and variable definitions:

Formal parameters for the *huffman_decode* routine are given in the following table:

Argument	Type	Description
imat	matrix *	Pointer to input matrix struct
omat	matrix *	Pointer to output matrix struct

The routine *huffman_decode* returns a type integer status value which is TRUE (1) if successful, or FALSE (0) if an error occurred.

Other routines called:

none

4.3.15 arith_encode

Purpose:

The library routine *arith_encode* performs data compression using an arithmetic coding algorithm.

Process:

The routine *arith_encode* accesses the data using pointers in the memory matrix structs that are passed to the routine. The memory matrix struct type was described in a previous section.

Interfaces and variable definitions:

Formal parameters for the *arith_encode* routine are given in the following table:

Argument	Type	Description
imat	matrix *	Pointer to input matrix struct
omat	matrix *	Pointer to output matrix struct

The routine *arith_encode* returns a type integer status value which is TRUE (1) if successful, or FALSE (0) if an error occurred.

Other routines called:

none

4.3.16 arith_decode

Purpose:

The library routine *arith_decode* decompresses data that was previously compressed using the arithmetic coding algorithm utilized in the routine *arith_encode*.

Process:

The routine *arith_decode* accesses the data using pointers elements of the memory matrix structs that are passed to the routine.

Interfaces and variable definitions:

Formal parameters for the *arith_decode* routine are given in the following table:

Argument	Type	Description
imat	matrix *	Pointer to input matrix struct
omat	matrix *	Pointer to output matrix struct

The routine *arith_decode* returns a type integer status value which is TRUE (1) if successful, or FALSE (0) if an error occurred.

Other routines called:

none

5 Test Plan

Two types of tests will be performed on the various software modules, (1) validation test suites, and (2) acceptance tests.

5.1 Validation Test Suites

Validation test suites will be created for each kroutine using the Khoros non-interactive test suite generation infrastructure, described in Chapter 7 of the *Khoros Toolbox Programming Guide.*

The purpose of these test suites is to provide a reproducible, non-interactive method of algorithm verification that helps guarantee the integrity, robustness, and portability of the code. This is particularly useful when the kroutines are recompiled on a new machine architecture.

A test shell script will be written for each kroutine based upon a test script template provided with the Khoros software distribution.

Small data files will be generated within or loaded from these shell scripts to provide input file stimuli for the tests. There will be no dependencies of these test scripts on other toolboxes except those that are part of the Khoros test suite development infrastructure.

These shell scripts will be located in a *testsuites* subdirectory to be placed under the data compression toolbox directory.

The standalone C-language versions of these kroutines will be developed first and then ported to the Khoros environment. A number of informal tests will be performed on the standalone versions before they are ported to Khoros. Since they will be fully tested under the Khoros environment, the validation test suites developed under the Khoros environment will not be duplicated for the standalone versions upon which they are derived.

5.2 Acceptance Tests

The following tests will be performed using the small number of RSTER data files that have been been made available to us for development testing:

(1) A Cantata workspace will be constructed that reads and compresses the RSTER data using the RSTERcompress kroutine, then uncompresses the just-compressed RSTER data using the RSTERuncompress kroutine. Both the original uncompressed data and the compressed-then-uncompressed data will be compared to demonstrate that no information has been changed or lost in the compress-decompress process. The sizes of the original RSTER files and their compressed versions will also be compared to demonstrate in an average sense across all the files, that compression has occurred.

(2) The standalone versions of the RSTER compress and uncompress routines will be demonstrated by command line execution to show that no information is changed or lost in the compress-decompress process, and by examination of file sizes, that compression has occurred in an average sense across all the files.

The following tests will be performed using a small number of SAR files obtained from the ADTS sensor rural clutter collection, and converted to Matlab-compatible file format:

(1) A Cantata workspace will be constructed that reads and compresses the SAR data using the SARcompress kroutine, then uncompresses the just-compressed SAR data using the SARuncompress kroutine. Both the original uncompressed data and the compressed-then-uncompressed data will be compared to demonstrate that no information has been changed or lost in the compress-decompress process. This will be demonstrated for both the Huffman and arithmetic coding algorithms. The sizes of the original SAR files and their compressed versions will also be compared to demonstrate in an average sense across all the files, that compression has occurred.

(2) The standalone versions of the SAR compress and uncompress routines will be demonstrated by command line execution to show that no information is changed or lost in the compress-decompress process, and by examination of file sizes, that compression has occurred in an average sense across all the files.

10. RLSTAP_HPC Data Compression Effort

Brendan Bradley
Albuquerque High Performance Computing Center (AHPCC)

30 September, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

Outline

- Introduction
- Background
  - Compression algorithms
  - Available data
- System design
  - Algorithm selection
  - Software tools
- Compression results
  - RSTER data compression
  - SAR data compression

Objectives

- Add *lossless* data compression capabilities to the RLSTAP software environment for Air Early Warning (AEW) radar simulation.
- Provide the ability to efficiently compress two types of data
  - RSTER radar data
  - Formed SAR images
- Provide standalone versions of the data compression software tools

Challenges

- Data compression must be lossless which eliminates a number of techniques capable of achieving much higher levels of compression
- Both RSTER data and SAR data are random and nondeterminstic with fairly broad probability densities
- Utilize non-proprietary techniques and software that can be incorporated into the RLSTAP environment without constraint

Outline

- Introduction
- Background
  - Compression algorithms
  - Available data
- System design
  - Algorithm selection
  - Software tools
- Compression results
  - RSTER data compression
  - SAR data compression

Objectives

- Add *lossless* data compression capabilities to the RLSTAP software environment for Air Early Warning (AEW) radar simulation.
- Provide the ability to efficiently compress two types of data
  - RSTER radar data
  - Formed SAR images
- Provide standalone versions of the data compression software tools

Challenges

- Data compression must be lossless which eliminates a number of techniques capable of achieving much higher levels of compression
- Both RSTER data and SAR data are random and nondeterminstic with fairly broad probability densities
- Utilize non-proprietary techniques and software that can be incorporated into the RLSTAP environment without constraint

Outline

- Introduction
- **Background**
  - Compression algorithms
  - Available data
- System design
  - Algorithm selection
  - Software tools
- Compression results
  - RSTER data compression
  - SAR data compression

Overview of lossless compression techniques

Most lossless compression techniques fall into one of the following categories:

- Statistical methods
  - idea: Estimate probabilities of symbols and assign symbols with high probability to shorter codewords
  - example: Huffman coding
- Dictionary methods
  - idea: Replace "words" or fragments of words with an index to an entry in a dictionary
  - example: Lempel-Ziv coding

Several lossless algorithm schemes were investigated

Arithmetic:
- A class of coding schemes by which it is possible to encode in non-integer bits. This allows us, in theory, to code close to entropy

Huffman:
- Works on same general principle as arithmetic coding but codes are restricted to integer lengths

Lempel-Ziv:
- These algorithms are, in essence, adapted dictionary routines

GZIP:
- A variant of LZ77 coding which uses an offset pointer and length pointer to point to previously occurring section of "text"

Compress:
- A variant of LZ78 which is similar to LZ77 but with restrictions placed on which substrings may be referenced

Additional algorithms that were investigated

Prediction by Patial Matching:	• Uses finite-context models of symbols which are then encoded by a statistical, usually arithmetic, encoder
Dynamic Markov Compression:	• Based on a finite state machine. The coding is adaptive so that both probabilities and structure of finite state machine changes as coding proceeds
Ad hoc schemes:	• Ways to exploit unique characteristics of the data and how it is archived - tricks

RSTER data

- Complex data composed of integer real and imaginary parts each representable in no more than 24-bits
- Histograms of real and imaginary parts resemble Gaussian pdfs although some skewness is observed
- Data arranged in a three-dimensional cube with axes representing range, pulse and elements (RPE)
- Header information and data cube are stored as Matlab 4.2c - compatible matrix file
  - data cube is stored in a complex matrix
  - each header attribute is stored in its own matrix
  - data cube data type in file either short integer or long integer

SAR data

- Customer data not available - utilized formed SAR images of clutter from MSTAR-CD (public domain, non ITAR)
  - rural and urban clutter
  - h-h polarization
  - 15 degree depression angle
- MSTAR-CD data consists of textual "PHOENIX" header followed by binary "C4PL" header followed by 16-bit scaled magnitude values, followed by 12-bit scaled phase values stored in 16-bits.
- Data converted to 16-bit I and 16-bit Q values for data compression experiments

Outline

- Introduction
- Background
  - Compression algorithms
  - Available data
- **System design**
  - Algorithm selection
  - Software tools
- Compression results
  - RSTER data compression
  - SAR data compression

Initial algorithm downselection

- SAR data was not available early on in the development, so the initial focus of the work was on RSTER data
- After an initial examination of the RSTER data set we opted for
  arithmetic: optimal compression in the sense of entropy
  Huffman: very robust, relies on probabilities only, not ordering
- The work of other researchers indicated that these schemes would also work well for SAR data in the formed complex domain
- Further investigation of the RSTER data revealed a storage inefficiency in the file format. This led to the development of an *ad hoc* technique applicable only to the RSTER database in its current form.

Ad hoc compression of RSTER matrix files

- Most of the storage space of RSTER files are due to the RPE data cubes rather than ancillary attributes
- When such data are stored to a file, Matlab recognizes that the data values are integral and attempts to store each datum as an integer whose precision is large enough to hold the array extrema. Thus, if only one value in the array exceeds the representable range of a two-byte integer, the storage type for the whole array defaults to four-byte integer.
- Our algorithm places entries of the data cubes that have values out of range (either 2 or 3 byte) to an ancillary array and flags the offending entries with a reserved value within range. This permits a high percentage of the data to be stored in short integers
- Decompression converts data to a larger integer type and replaces the flagged entries with their proper values from the ancillary array.

Availability of compression algorithms and software

- The ad hoc technique was developed in-house and software tools were constructed to compress and uncompress RSTER data
- Source code was obtained for the Huffman compression scheme which has been incorporated into software tools to compress and uncompress data.
- An extensive search was conducted for public-domain arithmetic coding software but all were subject to severe copyright restrictions.
- An algorithm alternative, GZIP, was sought as a backup.
- Recently, an arithmetic coding package was located at Sandia Labs which is tuned to compress 16-bit I and Q SAR data. We were given an executable binary version with which we have been able to conduct experiments. We are still waiting to hear about the availability of source code.

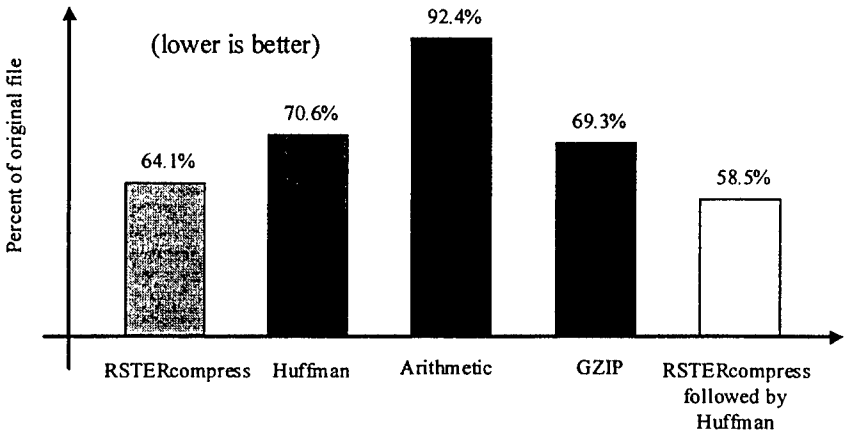
Data compression software tools (Khoros kroutines)

RSTERcompress	–	Utilizes the ad hoc technique to reduce the size of RSTER files
RSTERuncompress	–	Restores RSTER files to their original state
SARcompress	–	Uses Huffman algorithm to compress SAR data (can also be applied to RSTER data)
SARuncompress	–	Restores Huffman encoded data to original state
GZIPcompress	–	Uses GZIP algorithm to compress data
GZIPuncompress	–	Restores GZIP encoded data to original state

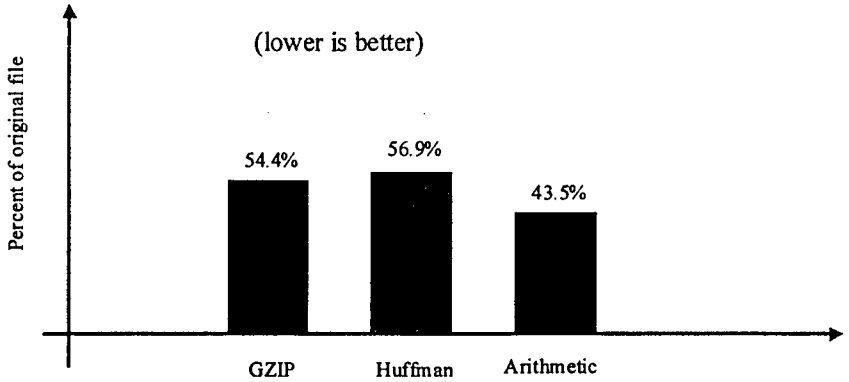
Outline

- Introduction
- Background
  - Compression algorithms
  - Available data
- System design
  - Algorithm selection
  - Software tools
- **Compression results**
  - RSTER data compression
  - SAR data compression

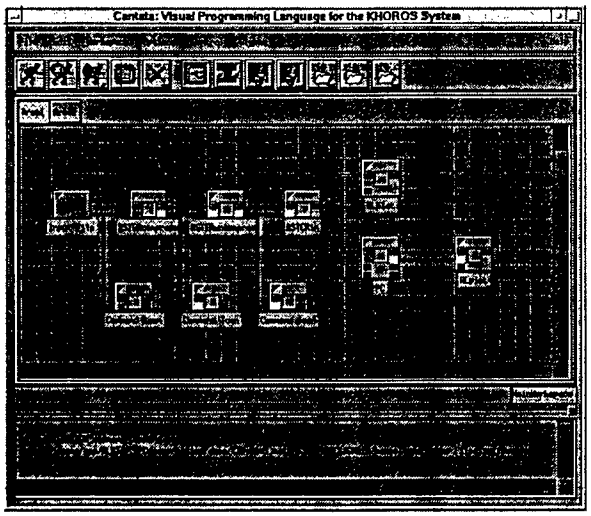
Results of data compression on available RSTER data



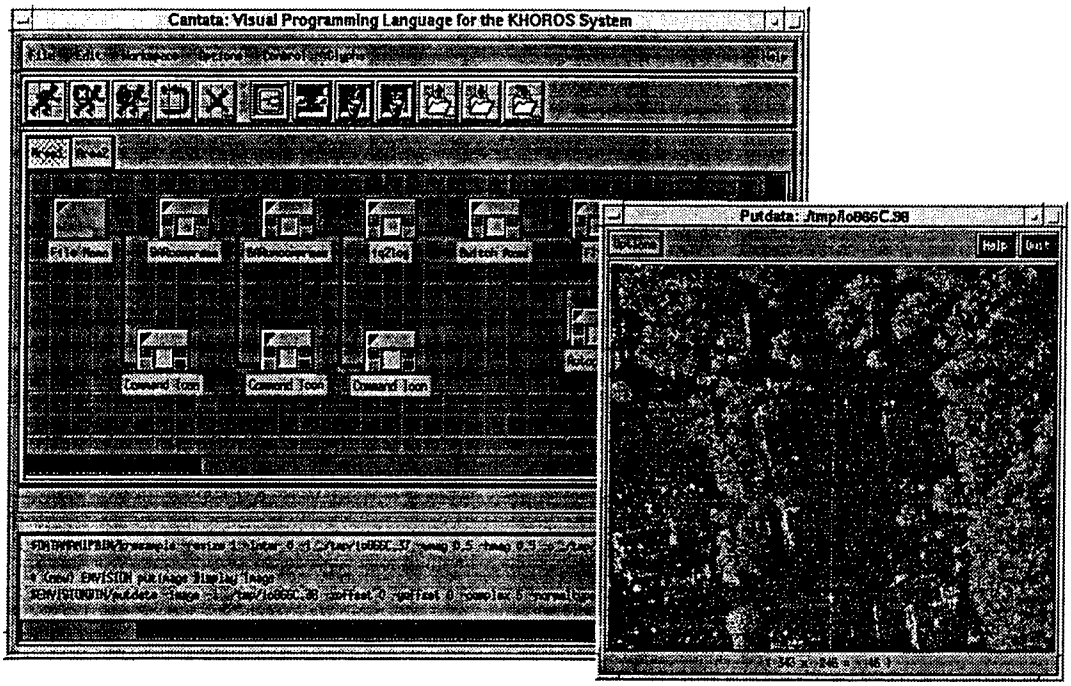
Results of data compression on available SAR data



Example RSTER data compression workspace



Example SAR data compression workspace



Summary

- Lossless data compression algorithms were investigated for RSTER and SAR data
- Both statistical and dictionary based lossless schemes proved useful
- The best single-approach compression results for RSTER data in its current storage format was achieved using an *ad hoc* technique. This technique followed by Huffman coding produced the best overall results
- The best overall compression results for SAR data were achieved using an arithmetic coding scheme. GZIP and Huffman also did quite well
- Software tools were developed that utilize Huffman, GZIP and the *ad hoc* algorithm. Arithmetic coding software not yet available.

11. Compression Manual

Brendan Bradley
Albuquerque High Performance Computing Center (AHPCC)

30 October, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

Introduction to the Khoros RSTER Routines

A. Overview

With large sets of RSTER (Radar Surveillance Technology Experimental Radar) -like data archived at MHPCC in Kihei, Hawaii and researchers located at diverse geographic locations throughout the country, there is a frequent need to distribute data from the archive in Maui to distant researchers. The primary modality for this details the ethernet. The combination of large data sets and limited communication bandwidth drive a need for data compression for such file transfers. Lossless compression is required to ensure that algorithm performance validation can be performed accurately. This toolbox is designed for this purpose.

Each RSTER matrix file is composed of one or more "cpi" matrices, and a number of ancillary arrays and scalar values, all stored as a sequence of Matlab version 4.2 matrices. A Matlab version 4.2 matrix file contains one or more matrices each consisting of a 20-byte header, followed by a matrix name string, followed by actual matrix data. Each matrix is stored sequentially and contiguously in the file.

The Matlab header consists of five 4-byte signed integers that define, in order, (1) the matrix type, (2) number of rows in the matrix, (3) number of columns in the matrix, (4) whether the matrix is real or complex, and (5) the length of the matrix name including a NULL terminator character. The matrix type encodes the precision and the data type of the matrix, the machine architecture upon which the data was generated, and whether the matrix is sparse, numeric, or contains text. All matrices in a RSTER matrix file are either numeric or textual.

All matrices are stored as type double (real or complex) data values in memory within Matlab. However, to reduce storage requirements when large matrices are stored to files using the Matlab "save" command, data is converted to a data type that requires fewer bytes-per-item where possible according to an internal algorithm. For example, if all data within a real type double matrix are integral values (i.e. representable as integers), and are bounded by the representable range of 32-bit signed integers, the data is converted to and stored as 32-bit signed integers automatically when saved. If all data are integral and bounded by the representable range of 16-bit signed integers, the data is converted to and stored as 16-bit signed integers. Complex data, are similarly converted when real and imaginary components are tested independently against the bounds.

The "cpi" arrays, which constitute a high percentage of the storage requirements of a RSTER file, are complex data with real and imaginary components stored as signed integers. The Matlab format places all the real components of a matrix first in the file followed by all its imaginary components.

In most RSTER files the "cpi#" matrices are stored as 4-byte integers. When a high percentage of the data within the "cpi#" matrices have values that fall within the range -(2^15)+1 to (2^15)-1, that is, -32767 to +32767, the data that fall within this range are stored as 2-byte integers, and the out-of-range data are stored in ancillary arrays as 4-byte integers. This yields a compression factor that approaches 2:1. The Matlab "O" variable in the "type MOPT" flag is used to describe the type of compression used, called "ctype", and is given the value 1 to indicate the "cpi#" has been compressed to a 2-byte range with the out of range values being sent to the matrix "i_cpi#" immediately following the 2-byte "cpi#".

When a low percentage of the data within the "cpi#" matrices have values that fall within the range -(2^15)+1 to (2^15)-1, but a high percentage of the data fall within the range -(2^23)+1 to (2^23)-1, the data that fall within this range are stored as 3-byte integers. If in "cpi#" there exists values out of the 3-byte range then

these are stored in the index array "i_cpi#" and ctype = 2. If no values are out of the 3-byte range then there is no need for an index array and ctype = 4.

If a low percentage of the data within the "cpi#" matrices have values that fall within the range -(2^23)+1 to -(2^23)+1, or the "cpi#" matrices are not initially 4-byte integers, then no compression is performed, and input file is simply copied to the output file without modification. If no compression is performed, then ctype = 0.

Upon decompression, Matrices in the input file having names of the form "cpi#" are singled out. These matrices contain complex data types whose real and imaginary components are stored as either two-byte or three-byte integers, depending on the mode of compression performed. Once it has been confirmed that the matrix is a "cpi#" matrix then the "ctype" is extracted and the matrix is decompressed appropriately. For example if ctype = 1 or 2 then the "i_cpi#" following is recombined with the "cpi#". If ctype = 0, then the matric is output as is since no compression was performed in the first place.

Upon decompression the matrices in the files are bit by bit the same as the originals. i.e. the compression has been lossless.

A.1. RSTERcompress — *Compress RSTER data.*

Object Information

Category:	*COMPRESSION*	Subcategory:	*utilities*
Operator:	*RSTERcompress*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The kroutine RSTERcompress inputs a RSTER Matlab file, compresses the data in the file, and outputs a compressed RSTER file.

Command Line Arguments

```
Usage for RSTERcompress: Compress RSTER data.
% RSTERcompress
  -i          (infile)  RSTER matrix input file name
  -o          (outfile) Compressed output file name
```

A.2. RSTERuncompress — *Uncompress RSTER data.*

Object Information

Category:	*COMPRESSION*	Subcategory:	*utilities*
Operator:	*RSTERuncompress*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The kroutine RSTERuncompress inputs a compressed RSTER file, uncompresses the data in the file, and outputs a Matlab RSTER file.

Command Line Arguments

```
Usage for RSTERuncompress: Uncompress RSTER data.
% RSTERuncompress
  -i          (infile)  compressed RSTER input data file
  -o          (outfile) Uncompressed RSTER output data file.
```

A.3. kmat2asc — *Convert Matlab matrix data to Ascii.*

Object Information

Category:	*COMPRESSION*	Subcategory:	*utilities*
Operator:	*Mat2Ascii*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The kroutine kmat2asc reads each matrix in a Matlab matrix file and converts each datum to ascii in succession.

Command Line Arguments

```
Usage for kmat2asc: Convert Matlab matrix data to Ascii.
% kmat2asc
  -i          (infile)  Matlab matrix input file

  [-text]     (flag)    output ASCII text flag
  [-o]        (outfile) Ascii output file
              (default = (none))
```

B. Object Manifest

COMPRESSION toolbox Object Manifest			
Category Subcategory	**Operator**	**Description**	**Executable**
COMPRESSION	Extract CPI	Extract cpi array and place in value segment.	kextractcpi
utilities	Mat2Ascii	Convert Matlab matrix data to Ascii.	kmat2asc

COMPRESSION toolbox Object Manifest			
Category Subcategory	**Operator**	**Description**	**Executable**
	RSTERcompress	Compress RSTER data.	`RSTERcompress`
	RSTERuncompress	Uncompress RSTER data.	`RSTERuncompress`

B.1. lRSTERcompress() — *perform data compression on matrix file*

Synopsis

```
int lRSTERcompress(
  kfile *fdi,
  kfile *fdo)
```

Input Arguments

`fdi`

input file stream pointer

`fdo`

output file stream pointer

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This program inputs a Matlab version 4.x matrix file containing RSTER type data, compresses selected matrices within the file, and writes each matrix to a second Matlab version 4.x matrix file. If the Matlab file contains data in a supported machine format, it is converted to the local machine format as needed. Otherwise an error is reported.

Matrices in the input file having names of the form "cpi#", where "#" is a positive integer, are singled out for compression. These matrices contain complex data whose real and imaginary components are stored as integers. In each "cpi#" matrix, the real components are stored first, followed by all the imaginary components.

In most RSTER files the "cpi#" matrices are stored as 4-byte integers. When a high percentage of the data within the "cpi#" matrices have values that fall within the range -(2^15)+1 to (2^15)-1, that is, -32767 to +32767, the data that fall within this range are stored as 2-byte integers, and the out-of-range data are stored in ancillary arrays as 4-byte integers. This yields a compression factor that approaches 2:1. The Matlab "O" variable in the "type MOPT" flag is used to describe the type of compression used, called "ctype", and is given the value 1 to indicate the "cpi#" has been compressed to a 2-byte range with the out of range values being sent to the matrix "i_cpi#" immediately following the 2-byte "cpi#".

When a low percentage of the data within the "cpi#" matrices have values that fall within the range -(2^15)+1 to (2^15)-1, but a high percentage of the data fall within the range -(2^23)+1 to (2^23)-1, the

data that fall within this range are stored as 3-byte integers. If in "cpi#" there exists values out of the 3-byte range then these are stored in the index array "i_cpi#" and ctype = 2. If no values are out of the 3-byte range then there is no need for an index array and ctype = 4.

If a low percentage of the data within the "cpi#" matrices have values that fall within the range -(2^23)+1 to -(2^23)+1, or the "cpi#" matrices are not initially 4-byte integers, then no compression is performed, and input file is simply copied to the output file without modification. If no compression is performed, then ctype = 0.

The following lists the possible compression modes:

0 CTYPE_NORMAL no compression 1 CTYPE_INT2I 2-byte compression with index array 2 CTYPE_INT3I 3-byte compression with index array 3 CTYPE_INT2N 2-byte compression w/o index array 4 CTYPE_INT3N 3-byte compression w/o index array

Restrictions

The input file must be a Matlab 4.x matrix type

B.1.1. lRSTERuncompress() — *uncompress RSTER data file*

Synopsis

```
int lRSTERuncompress(
  kfile *fdi,
  kfile *fdo)
```

Input Arguments

`fdi`
: input file stream pointer

`fdo`
: output file stream pointer

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This program inputs a Matlab version 4.x matrix file containing compressed RSTER type data, uncompresses selected matrices within the file, and writes each matrix to a second Matlab version 4.x matrix file. If the Matlab file contains data in a supported machine format, it is converted to the local machine format as needed. Otherwise an error is reported.

Matrices in the input file having names of the form "cpi#" are singled out for decompression. These matrices contain complex data types whose real and imaginary components are stored as either two-byte or three-byte integers, depending on the mode of compression performed. Once it has been

comfirmed that the matrix is a "cpi#" matrix then the "ctype" is extracted and the matirx is decompressed appropriately. For example if ctype = 1 or 2 then the "i_cpi#" following is recombined with the "cpi#". If ctype = 0, then the matric is output as is since no compression was performed in the first place.

Upon decompression the matrices in the files are bit by bit the same as the originals. i.e. the compression has been lossless.

B.1.2. lcalc_statistics() — *calculate word length statistics*

Synopsis

```
void
lcalc_statistics(
  matrix *mat,
  int *count2,
  int *count3)
```

Input Arguments

`mat`
: pointer to matrix structure

Output Arguments

`count2`
: count of integers outside of 2-byte range

`count3`
: count of integers outside of 3-byte range

Description

This procedure counts the number of entries in a matrix that lie out of the representable range of 2- and 3-byte signed integers. If the matrix is empty, zero counts are returned.

B.1.3. lkmat2asc() — *convert matrix file data to ascii*

Synopsis

```
int lkmat2asc(
  kfile *fdi,
  int do_text,
  kfile *fdo)
```

Input Arguments

`fdi`
input file stream pointer

`do_text`
TRUE if data is to be sent to stdout FALSE if not.

`fdo`
output file stream pointer, or NULL

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure read each matrix in a Matlab matrix file and converts each datum to ascii in succession. Each datum is separated by a "newline" character. If the data is complex, the real and imaginary components are listed on the same line, separated by a "space" character.

B.1.4. ltranspose() — *transpose a 2-D matrix of any type*

Synopsis

```
static unsigned char msk[] = {
  0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80
};

int ltranspose(
  int si,
  int *d1,
  int *d2,
  kaddr x)
```

Input Arguments

`si`
size of each matrix item in bytes int *n1 - pointer to size in fast dimension int *n2 - pointer to size in slow dimension kaddr

`x`
pointer to matrix data of any type

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure transposes a 2-D matrix of any fixed-size data type in-place using a permutation-by-cycle algorithm. The matrix size parameters n1,n2 are interchanged to reflect the new matrix shape.

Reference: "Computational Frameworks for the Fast Fourier Transform", by Charles Van Loan, (SIAM) 1992.

Although the algorithm performs the transpose in place, an auxiliary vector of boolean values is required for algorithm housekeeping. A type "unsigned char" array is employed in which individual bits are utilized as flags. If the matrix is 1xN or Nx1 (a vector) the auxiliary vector is not allocated since no data movement is needed.

Restrictions

If array of strings, each item must be same length

B.1.5. lfree_matrix() — *free a Matlab-like matrix structure*

Synopsis

```
void
lfree_matrix(
  matrix *mat)
```

Input Arguments

`mat`

pointer to matrix structure

Description

This procedure frees a Matlab-like matrix structure created by lcreate_matrix(). If the pointer to the matrix is NULL, no action is taken.

B.1.6. lbswap_long() — *reverse byte order of type long datum*

Synopsis

```
void
lbswap_long(
  long *datum)
```

Input Arguments

`datum`

pointer to type long datum

Description

This procedure reverses the byte order of a single type long datum. If the pointer is NULL, no action is

taken.

B.1.7. lbswap_matrix() — *reverse byte order of matrix data*

Synopsis

```
int
lbswap_matrix(
  matrix *mat)
```

Input Arguments

mat
: pointer to matrix structure

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure reverses the byte order of matrix data in place. The real and imaginary components of complex matrix data must be contiguous in memory, that is, all real components must come first followed by all the imaginary components.

B.1.8. lcreate_matrix() — *create and initialize matrix structure*

Synopsis

```
matrix *
lcreate_matrix(
  char *name,
  int m,
  int n,
  int mtype,
  int dtype,
  int ctype)
```

Input Arguments

name
: matrix name

m
: number of rows in matrix

n

number of columns in matrix

`mtype`
matrix type (e.g. MTYPE_REAL)

`dtype`
data type (e.g. DTYPE_FLOAT)

`ctype`
compression type (e.g. CTYPE_INT2I) argument3 - explanation

Returns

Pointer to matrix struct, or NULL if an error occurred.

Description

This procedure creates a matrix by allocating a matrix structure, initializing the matrix header attributes, and allocating memory for matrix data. The matrix should be deallocated using the routine lfree_matrix().

B.1.9. ldtype_size() — *return size of matrix data type in bytes*

Synopsis

```
size_t
ldtype_size(
  int dtype)
```

Input Arguments

`dtype`
matrix data type (e.g. DTYPE_DOUBLE)

Returns

Size of data type in bytes, or zero if unsupported type

Description

This procedure returns the size in bytes of a matrix data type. If the type is unsupported, zero is returned. Only real data types are represented here. A matrix is complex if the matrix type is MTYPE_COMPLEX, regardless of the data type.

B.1.10. lfget_matrix_data() — *read matrix data segment*

Synopsis

```
int
lfget_matrix_data(
  matrix *mat,
  kfile *fd)
```

Input Arguments

`mat`
: pointer to matrix structure

`fd`
: file stream pointer

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure obtains the matrix size from the matrix header, allocates a data buffer for the data, reads in the matrix data, converts the data to local machine format, and updates the appropriate pointer entries in the matrix header struct.

If an error occurred, the data buffer is freed, the ointers in the matrix header struct are set to NULL, and the procedure returns FALSE (0). If successful, the procedure returns TRUE (1).

B.1.11. lfget_matrix_header() — *read in next matrix header*

Synopsis

```
matrix *
lfget_matrix_header(
  kfile *fd)
```

Input Arguments

`fd`
: file stream descriptor

Returns

Pointer to matrix structure, or NULL if an error

Description

This procedure allocates a matrix structure and reads a matrix header starting at the current file position pointer. Be sure to free the matrix structure when it is no longer needed, using the procedure lfree_matrix().

B.1.12. lfget_next_matrix() — *read in next matrix from transport*

Synopsis

```
matrix *
lfget_next_matrix(
  kfile *fd)
```

Input Arguments

`fd`
: file stream pointer

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure calls a sequence of two procedures that allocate a matrix structure, read a matrix header starting at the current file position pointer, allocate a buffer for the data based on the size information in the header, then read the matrix data that follows the header into the allocated buffer. If the machine format of the data in the file differs from the machine format of the local machine, the byte order of the data is reversed in place.

B.1.13. lfput_matrix_data() — *write out matrix data*

Synopsis

```
int
lfput_matrix_data(
  matrix *mat,
  kfile *fd)
```

Input Arguments

`mat`
: pointer to matrix structure

`fd`
: file stream pointer

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure writes out the matrix data from the buffer pointed to by the matrix struct to a matrix file

starting at the current file position.

B.1.14. lfput_matrix_header() — *write out matrix header*

Synopsis

```
int
lfput_matrix_header(
  matrix *mat,
  kfile *fd)
```

Input Arguments

`mat`
: pointer to matrix structure

`fd`
: file stream pointer

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure writes a file matrix header (struct Fmatrix) followed by the NULL-terminated matrix name to a matrix file starting at the current file position pointer.

B.1.15. lfput_next_matrix() — *write out matrix to transport*

Synopsis

```
int
lfput_next_matrix(
  matrix *mat,
  kfile *fd)
```

Input Arguments

`mat`
: pointer to matrix structure

`fd`
: file stream pointer

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure calls a sequence of two procedures that extract matrix information from a memory matrix struct (struct matrix) and write a file matrix (struct Fmatrix) header, a NULL-terminated matrix name, and finally, the matrix data, to a matrix file, beginning at the current file position.

B.1.16. lmmach_to_local() — *convert machine data order to local*

Synopsis

```
int
lmmach_to_local(
  matrix *mat)
```

Input Arguments

`mat`
pointer to matrix struct

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

This procedure converts memory matrix data from the machine format of the originating architecture to the machine format of the local architecture. If the machine architecture is not supported, no action is taken, and an error condition is returned. If successful, the "mat->mmach" entry in the matrix struct is updated to the value of the local architecture (MMACH_THIS_MACHINE).

The matrix namelength and dimensions are assumed to be in local machine format, which is done by the routine fget_next_header().

If successful, TRUE (1) is returned, otherwise FALSE (0) is returned. The user can also compare the resulting machine type in "mat->mmach" with MMACH_THIS_MACHINE to determine if the conversion was successful.

12. Critical Design Review For Advanced Signal Processing, RLSTAP Validation in Khoros 2.1

Marc Friedman
Maui High Performance Computing Center (MHPCC)
Joe Fogler
Albuquerque High Performance Computing Center (AHPCC)

6 May, 1997

To Support Contract Statement of Work Subtask 4.1.4.1, Research necessary changes to RLSTAP to migrate the SP-2 to Khoros 2.0 Interface.

1 Introduction

This report summarizes a comparative test and evaluation of selected software components of the Khoros 2.1 version of the Rome Laboratory Space Time Adaptive Processing / Algorithm Development Tool (RLSTAP/ADT) against the original Khoros 1.0 implementation.

The RLSTAP/ADT environment utilizes Khoros and its graphical programming environment Cantata to represent a state-of-the-art development environment for clutter modeling and radar simulation for advanced early warning (AEW) applications. Written initially in Khoros version 1.0, (RLSTAP_K1), RLSTAP has been ported to Khoros release, version 2.1, (RLSTAP_K2) by United States Research Lab - Rome with development performed by Kaman Sciences Inc. and Technology Services Corp.

RLSTAP V1.0 was sponsored by the Defense Advanced Research Projects Agency / Sensor Technology Office (DARPA/STO) Advanced Signal Processing (ASP) program which focuses on advanced processing techniques and technologies for next generation airborne early warning (AEW) platforms.[1] A key technology area for this activity is software tools and methodologies for collaborative algorithm development. RLSTAP V2 was sponsored by the DARPA/ Information Technology Office (ITO) to support foliage penetration technology.

Khoros and Cantata are the flagship products of Khoral Research Incorporated (KRI), a spinoff company from the University of New Mexico Electrical and Computer Engineering Department. Khoros is a software integration and development environment for information processing, data exploration and visualization.[2] Khoros is a comprehensive software system with a rich set of tools usable by both end-users and application developers. Included in these tools is Cantata which provides users the ability to construct complex algorithms by interconnecting iconic representations, called glyphs, of processing functions in a terminal window called a workspace, using mouse point-and-click operations. Khoros has become a de-facto standard for collaborative algorithm development in the Department of Defense automatic target recognition (ATR) community.

2 Scope

2.1 System Overview

The vast majority of programs written for or supplied with the Khoros environment are *kroutines*. These include all data processing programs such as image processing and signal processing routines. Kroutines are fully usable from within the Cantata graphical programming environment, but generally do not display graphics or images; they simply input data, process it, and output results.

The software tested and evaluated under this task was in the form of Khoros 2.1 compatible kroutines most of which were grouped into functional Cantata workspaces. The major software modules tested under this task, representing key functionalities in selected RLSTAP toolboxes, were as follows:

```
rsterin - a routine for importing RSTER files into RLSTAP
convbf  - a workspace representing a conventional beamformer algorithm
jdo     - a workspace representing a joint-domain-optimal STAP algorithm
phymod  - a workspace representing a physical radar model
```

Each of these software modules have RLSTAP counterparts written in Khoros 1.0. The Khoros 2.1 implementations were evaluated against their Khoros 1.0 counterparts to determine:

[1] G. W. Titi, An Overview of the ARPA/NAVY Mountaintop Program, Proceedings IEEE Adaptive Antenna Systems Symposium, (1994).

[2] See KRI's website at *http://www.khoral.com/* for more details.

1. That the Khoros 2.1 implementation (RLSTAP_K2) produces the same results within machine precision tolerance as the Khoros 1.0 implementation (RLSTAP_K1).

2. The execution speed of RLSTAP_K2 relative to that of RLSTAP_K1.

2.2 Limitations

The Khoros 1.0 version of RLSTAP was developed under the SunOS/Solaris operating system on a SUN SPARC computer. To simplify testing procedures, and to allow timing benchmark comparisons, both versions were tested on the same SPARC machine under the Solaris operating system. Although a RLSTAP_K2 distribution was successfully compiled under the IBM AIX operating system, only the Solaris results are reported herein.

The version of RLSTAP_K2 available for testing under this task was a preliminary version (designated distribution #6). Some of its capabilities which were to be evaluated, were not yet fully functional as of June 30, 1997, the deadline established in the design document that defined the validation task. In particular, the physical model workspace (phymod.wksp) was not yet fully functional. Consequently, only mathematical comparisons were performed on working sections of the workspace, and timing benchmarks were not performed.

3 Reference Documents

On-line help and manual pages from both the Khoros 1.0 and Khoros 2.1 implementations of RLSTAP served as guides for determining the correct operation of the functionalities to be tested. In addition, a preliminary Programmers Guide was received from Kaman Sciences in the form of a formatted Khoros 2.1 toolbox manual. A number of phone calls were also made to Kaman Sciences Inc., which provided invaluable information and tireless assistance in getting the environments to run.

4 Design

4.1 Software Development Plan

Though this task did not develope RLSTAP, number of shell script were written to facilitate comparisons between results from the Khoros 1.0 and Khoros 2.1 RLSTAP versions. These scripts are not deliverables under the validation task.

The following scripts were utilized for testing:

```
rcheck    - A script for computing the extrema of the pointwise absolute
            difference of two data files. Due to a difference in the
            ordering of data dimensions between Khoros 1.0 and Khoros 2.1
            files, a feature was incorporated into the script that
            reoriented the dimensions of Khoros 1.0 files to match those
            of their Khoros 2.1 counterparts before the comparisons.
            This script compared the real and imaginary parts separately.

rstats    - A script for comparing the statistics of two data files.
            This script was utilized when the Khoros 1.0 and Khoros 2.1
            versions of a routine yielded data vectors of slightly
            different length. Since pointwise comparisons could not be
            done in such cases, the results were compared statistically,
            (i.e. the means and standard deviations were compared).
```

```
convbf.sh - A script for timing the individual routines in the Khoros 1.0
            version of the conventional beamformer baseline workspace.
            The unix routine TIMEX was utilized to time individual routines.

jdo.sh    - A script for timing the individual routines in the Khoros 1.0
            version of the joint-domain-optimal basline workspace.

note:     - The Khoros 2.1 versions of the conventional beamformer and
            joint-domain-optimal workspaces were timed using the
            QSERVER data logger feature of the RLSTAP\_K2 environment.
```

4.2 Data Description

A number of RSTER data files were made available by MHPCC for testing. The detailed results presented in this report were obtained using the file *t38-01v3.mat*. This Matlab 4.2 compatible file was imported into the RLSTAP_K1 and RLSTAP_K2 environments using the corresponding RSTERin routine. The first coherent pulse interval data cube in the Matlab file, e.g. t38-01v2.mat.cpi.1, was utilized in the testing of the *convbf* and *jdo* workspaces. The *phymod* workspace generated its own data and did not require the converted Matlab file as input.

5 Test Results

Testing was performed in stages: (1) test compilation of RLSTAP_K2 on both the SPARC and AIX architectures, (2) mathematical comparison of results of selected routines and workspaces in RLSTAP_K1 versus RLSTAP_K2, and (3) a comparison of the execution speed of the two implementations on the same machine architecture and configuration.

5.1 Test Compilation

The RLSTAP_K2 environment was re-compiled on an AIX based R6000 workstation.

```
The compilation procedures for RLSTAP under Khoros 2.1 on an AIX
machine were as follows:

(1) Create home directory for RLSTAP2 and tar in distribution tape
(2) Run distribution command file
    - chmod +x set.6.cmd   # assumes distribution number is six
    - set.6.cmd

(3) Uncompress the distribution archive
    - uncompress -c set.6.tar.Z | tar xvf

(4) For a full installation (distribution includes source code)
    - swtools/bin/r2bootstrap -makemakeall -install

Additional compilation procedures:

(5) Modify the defines in AIX.cf configuration file in Khoros
    bootstrap config directory as follows
    - #define FortranOptFlags -O -qextname
```

```
    - #define LibraryFOptFlags -O -qextname
    - #define FortranDebugFlags -g -qextname
    - #define LibraryFDebugFlags -g -qextname

(6) Set environmental variables as follows
    - FFLAGS="-O -qextname"
    - EXTRA_LOAD_FLAGS=-xlf90

These settings address trailing underscores in function names
and the linking of FORTRAN code in Makefiles
```

All toolboxes compiled under AIX except the *plot* toolbox. There were some missing Imakefiles and Makefiles in the higher directory levels of this toolbox. However, individual tools could be compiled under AIX manually from within their own subdirectories.

The plot tool *rlplot* would not compile successfully until a Macro name was redefined. In the file *rl_3d_scene.c* a macro named DTR was redefined as the name of a constant. By redefining the name of the constant to DTRAT (DT ratio) so that it would no longer conflict with the Macro name, the plot tool could then be compiled.

5.2 Environment Setup

Once the RLSTAP_K2 distribution was compiled, the user environment needed to be configured. The steps are as follows:

```
(1) Create a working directory in user area and go there

(2) Source the r2developer shell script to initialize environment
     % source $R2_HOME/swtools/bin/r2developer

(3) Run init_rlstap2 script to create working subdirectories
    for data workspaces. A script called rlstap2 is also created.
     % $R2_HOME/universal/bin/init_rlstap2

(4) Create a file called .Toolboxes in user's home directory
    (not the previously mentioned working directory). Add names and
    paths of personal toolboxes to be included with RLSTAP2 on invocation.

(5) To execute RLSTAP\_K2 from working directory, run rlstap2
    script in foreground.
     % rlstap2
```

5.3 Verification of Selected Routines

The following routines and workspaces were first executed in the RLSTAP_K1 environment to establish a baseline. The same routines and workspaces were then executed in the RLSTAP_K2 environment and the results generated by the RLSTAP_K2 environment were compared against those generated in the RLSTAP_K1 version. Any differences that were encountered are noted.

1. rsterin - a routine for importing RSTER files into RLSTAP
2. convbf - a workspace representing a conventional beamformer algorithm
3. jdo - a workspace representing a joint-domain-optimal STAP algorithm

4. phymod - a workspace representing a physical radar model

5.3.1 RSTERin

RSTER data in the form of Matlab 4.2 compatible matrix files is imported into the RLSTAP environment using *RSTERin*. Each matrix file may contain more than one coherent pulse interval (cpi) data cube, which is placed in its own individually numbered output file. In the RLSTAP_K2 version, the cpi data are placed in the value segment of a Khoros data object. Ancillary matrices (mostly scalars) are placed in global attributes of the Khoros data object.

The following data exchanges were tested in RLSTAP_K2:

- RSTER data into RLSTAP
- RLSTAP to RSTER data
- RLSTAP to Matlab
- Matlab to RLSTAP

RSTERin was found to function properly in each instance. The results of the Khoros 1.0 and Khoros 2.1 versions were compared using the aforemetioned *rcheck* script and were shown to be mathematically identical. However, two phenomena were noted:

(1) When Khoros 1.0 RSTER data was imported into Khoros 2.1 routines used to compare data values, the depth and elements dimensions in the Khoros 2.1 polymorphic data model were interchanged. Although this did not actually affect data ordering, the dimension labels had to be interchanged (using the Khoros {\it korient} routine) before subsequent data comparisons could be made.

(2) A number of the RLSTAP_K2 data object attribute names, even without their {\it rlstap2_} prefixes, were too long for Matlab 4.2 to handle and were truncated to 19 characters when converted back into Matlab form via the routine RSTERout.

5.3.2 convbf

The conventional beamformer RLSTAP_K2 workspace was composed of the following glyphs (routines):

- WindowGen (window data generator, e.g. Hamming) (4 instances)
- PC (pulse compression)
- Extract (extract section of data cube)
- MotionComp (motion compensation)
- MTI (moving target indicator)
- DopplerSubband
- FFT
- Normalize
- ConvRule (conventional beamformer rule)
- SteerVector
- CnvBF
- Floor
- CA_CFAR (cell averaging CFAR)
- GO_CFAR (greatest-of CFAR)
- OS_CFAR (ordered statistic CFAR)
- TM_CFAR (trimmed-mean CFAR)
- Detect (detection module) (4 instances)

The Khoros 1.0 and Khoros 2.1 versions of the *convbf* workspace are depicted in Figure 1 and Figure 2, respectively. When multiple instances of the same glyph (e.g. WindowGen) were encountered in a workspace, they were numbered sequentially starting with the top left of the workspace and proceeding toward the bottom right. The outputs of each glyph in the workspaces were given names that reflect the glyphs from which they were generated. For example, the output of the second Detect glyph was designated 'Detect2.kdf' in the Khoros 2.1 workspace. The workspace glyph pane parameters were adjusted in the Khoros 1.0 version to match the baseline workspace settings of the Khoros 2.1 version. Occasionally, settings in the Khoros 2.1 version were found to be inconsistent. In that event, information was derived from the Khoros 1.0 version to determine proper settings.

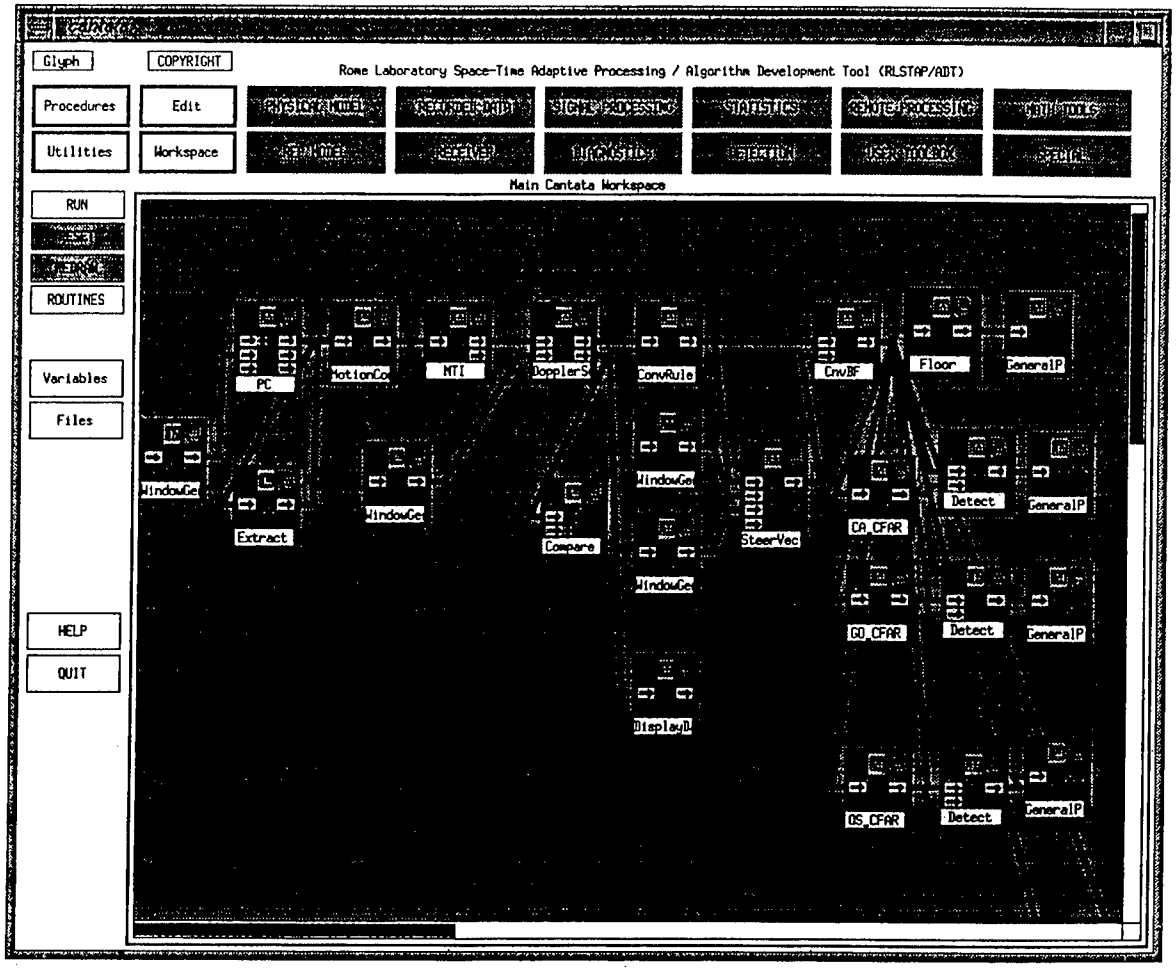


Figure 1. Khoros 1.0 conventional beamformer workspace.

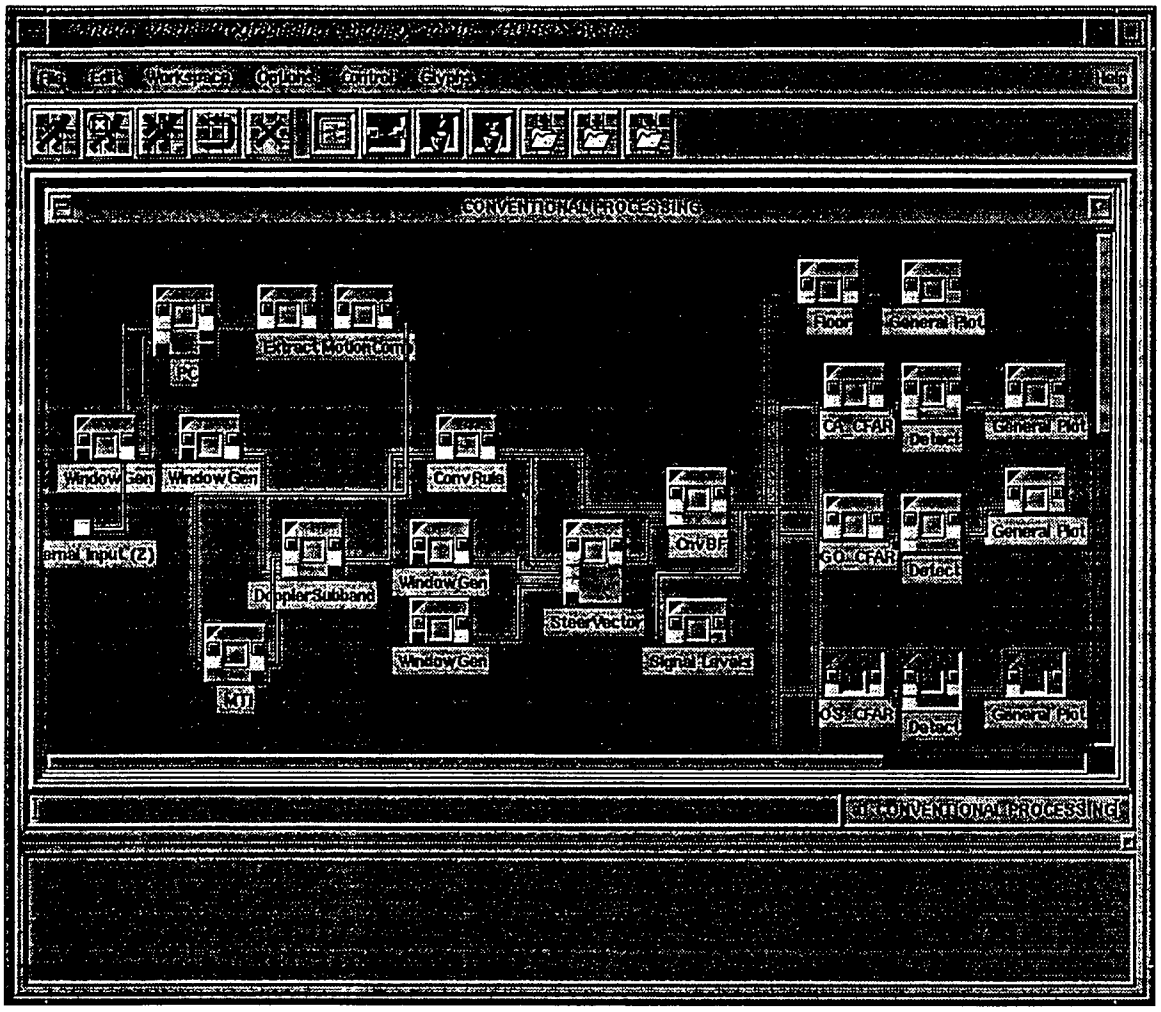


Figure 2. Khoros 2.1 conventional beamformer workspace.

Each computational glyph produced identical results in the RLSTAP_K1 and RLSTAP_K2 versions except for the following two glyphs:

Normalize:

The real and imaginary parts of the Khoros 1.0 and Khoros 2.1 versions differed very slightly (by one count out of 24 mantissa bits). Such differences can arise from performing computations in a slightly different order in the two cases. The difference is not significant.

Floor:

The Khoros 2.1 RLSTAP Floor function would not complete without error in this workspace. An error was reported from Khoros data services in the kdms layer, indicating that an index order had not been specified. This may have been caused by a memory leak but that is not certain. This same function produced the correct result in the jdo workspace and did not report an error. There may be some combination of settings that causes this failure.

5.3.3 Joint Domain Optimum STAP

The joint-domain-optimal STAP RLSTAP_K2 workspace was composed of the following glyphs (routines):

```
- WindowGen  (window data generator, e.g. Hamming)  (4 instances)
- PC         (pulse compression)
- Extract    (extract section of data cube)   (3 instances)
- MotionComp (motion compensation)
- MTI        (moving target indicator)
- JDORule    (joint-domain-optimal rule)
- Covar      (covariance)
- DiagLoad
- InvCovar   (inverse covariance)
- SteerVector
- STAPWgts
- BeamForm
- Floor
- SignalLevels (has no data object output)
- CA_CFAR    (cell averaging CFAR)
- GO_CFAR    (greatest-of CFAR)
- OS_CFAR    (ordered statistic CFAR)
- TM_CFAR    (trimmed mean CFAR)
- Detect
```

The Khoros 1.0 and Khoros 2.1 versions of the *jdo* workspace are depicted in Figure 3 and Figure 4, respectively. When multiple instances of the same glyph (e.g. WindowGen) were encountered in a workspace, they were numbered sequentially starting with the top left of the workspace and proceeding toward the bottom right. The outputs of each glyph in the workspaces were given names that reflect the glyphs from which they were generated. For example, the output of the second Detect glyph was designated 'Detect2.kdf' in the Khoros 2.1 workspace. The workspace glyph pane parameters were adjusted in the Khoros 1.0 version to match the baseline workspace settings of the Khoros 2.1 version. Occasionally, settings in the Khoros 2.1 version were found to be inconsistent. In that event, information was derived from the Khoros 1.0 version to determine proper settings.

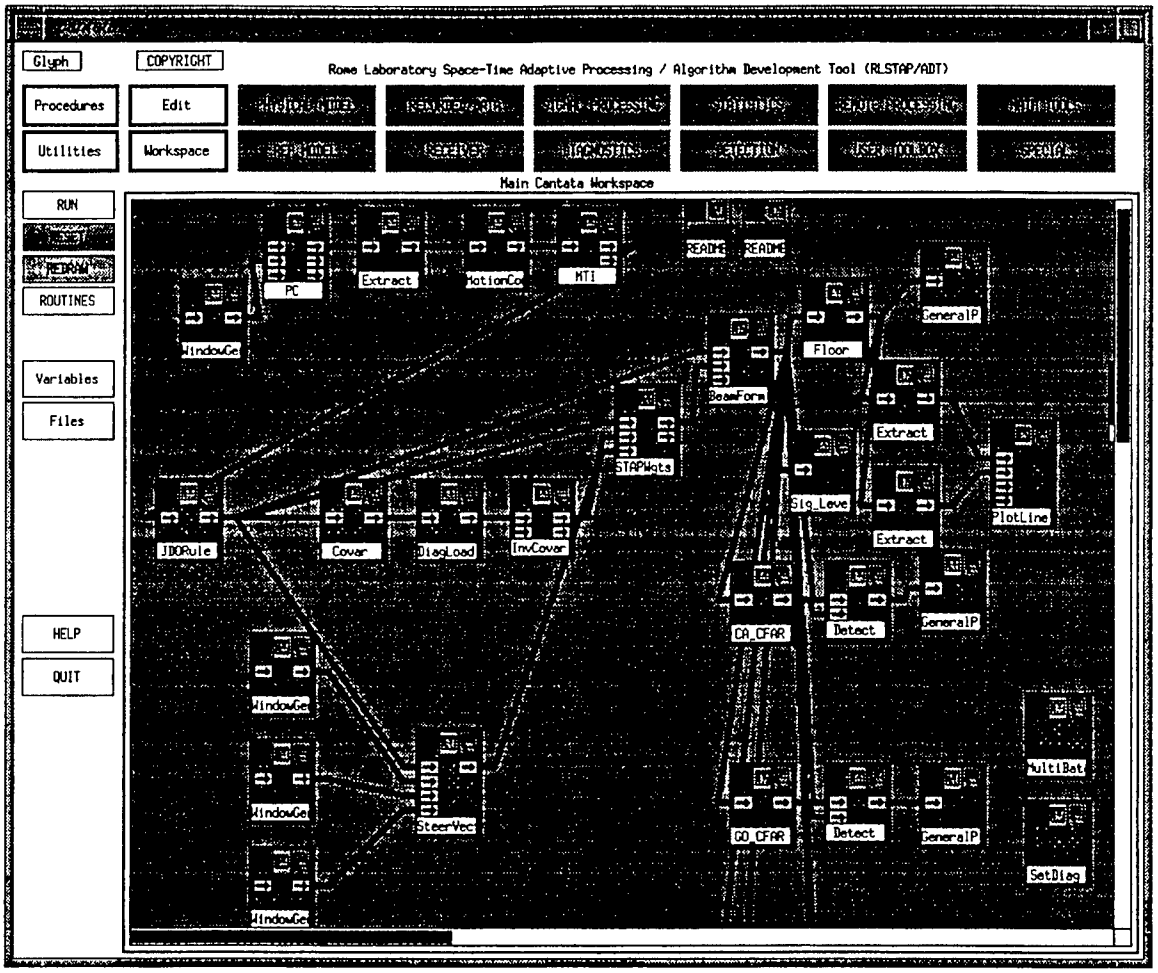


Figure 3. Khoros 1.0 joint-domain-optimal STAP workspace.

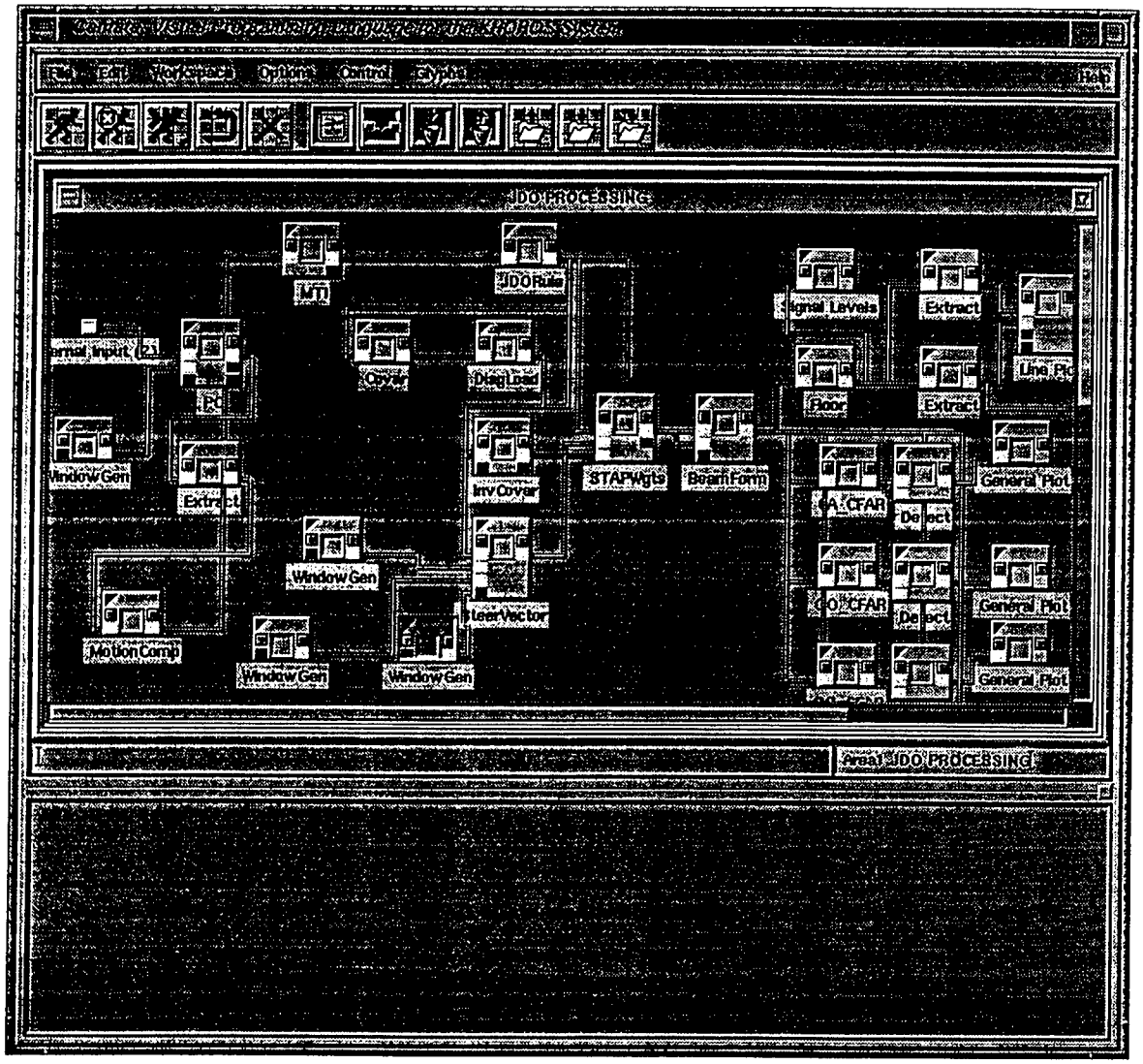


Figure 4. Khoros 2.1 joint-domain-optimal STAP workspace.

Each computational glyph in the jdo workspace produced identical results in the RLSTAP_K1 and RLSTAP_K2 versions except for the following minor differences:

Floor:
The Floor glyph outputs differed slightly in the Khoros 1.0 and Khoros 2.1 versions. This difference was only one count of 24 mantissa bits, and can be considered insignificant. However, based on results from the convbf baseline workspace, there are certain combinations of parameters and data that cause the glyph to fail. See the previous convbf section for further details.

Extract:
The outputs of Extract2 and Extract3 differed from their Khoros 1.0 counterparts because their input stimulus was the output of the Floor glyph. The differences did not exceed one count out of 24 mantissa bits. The first Extract (Extract1) glyph did not depend on the output of the Floor glyph, and produced identical results to its Khoros 1.0 counterpart.

5.3.4 phymod

The physical model RLSTAP_K2 workspace was composed of the following glyphs (routines):

```
- WindowGen  (window data generator, e.g. Hamming)  (3 instances)
- PMAntPatWin (antenna pattern window generator)    (3 instances)
- Normalize   (scale amplitude)                     (3 instances)
- PMSetup
- PMEnv
- RcvPlat
- AntPatCos  (antenna pattern cosine shape)
- RcvAnt
- XmtPlat
- XmtAnt
- Target    (target model)                          (2 instances)
- BNJam     (jammer model)                          (2 instances)
- CltrMaps
- CltrPlse
- RPESum
- RcvrNoise
- RcvrBP
- SignalLevels   (no data object output)
- CNRCalc
```

The Khoros 1.0 and Khoros 2.1 versions of the *phymod* workspace are depicted in Figure 5 and Figure 6, respectively. When multiple instances of the same glyph (e.g. WindowGen) were encountered in a workspace, they were numbered sequentially starting with the top left of the workspace and proceeding toward the bottom right. The outputs of each glyph in the workspaces were given names that reflect the glyphs from which they were generated.

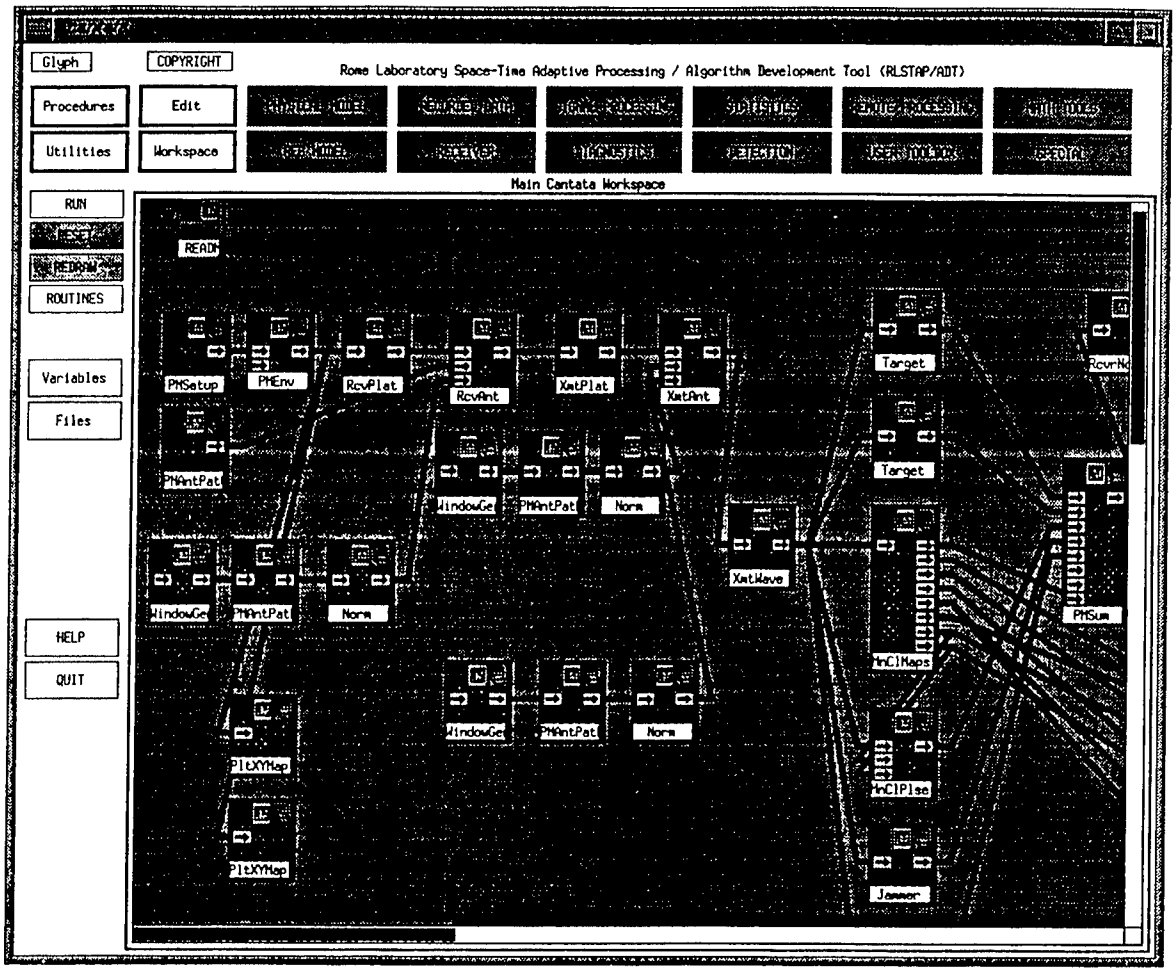


Figure 5. Khoros 1.0 physical model workspace.

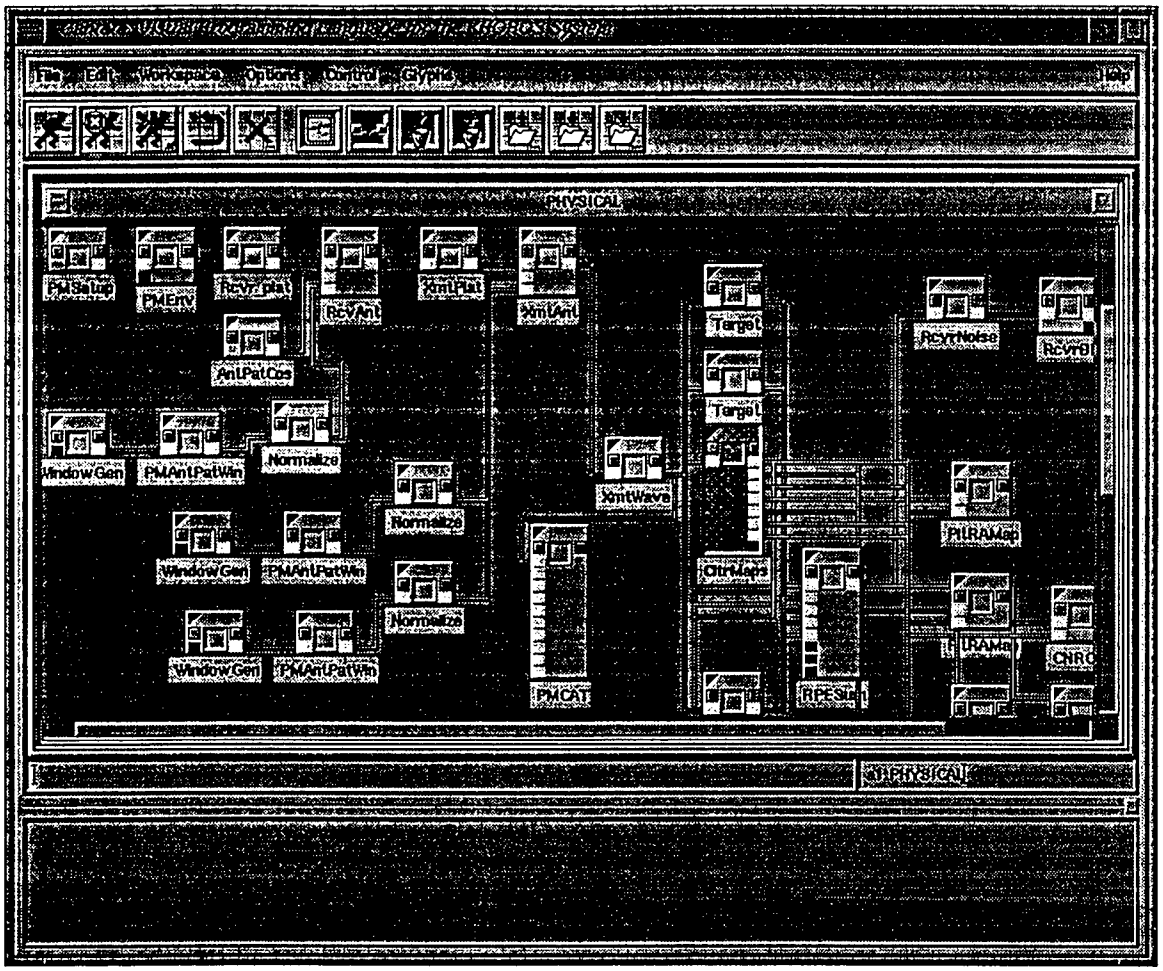


Figure 6. Khoros 2.1 physical model workspace.

The following phymod workspace glyphs produced results that differed between the RLSTAP_K1 and RLSTAP_K2 versions:

Normalize1, Normalize2, Normalize3:
The real and imaginary parts of the Khoros 1.0 and Khoros 2.1 versions differed very slightly (by one count out of 24 mantissa bits). Such differences can arise from performing computations in a slightly different order in the two cases. The difference is not significant.

PMEnv:
This glyph primarily generates attribute information. However, it also generates a value segment (in Khoros 2.1 version) that contains a single complex value. This value is apparently unused by succeeding glyphs which simply extract attribute information from this glyph's output. The data segments of both the Khoros 1.0 and Khoros 2.1 counterparts contain a single uninitialized value which is different in the two versions. Since the datum is not used by subsequent glyphs, its value is of no consequence.

RcvPlat:
This glyph primarily generates attribute information. It propagates the single datum from PMEnv to its own output, but does not appear to utilize the value in any calculations.

Rcvant, XmtPlat, and XmtAnt:
These glyphs generate data that are nearly identical to their Khoros 1.0 counterparts. The output data values only differ by a maximum of one count out of a maximum 24 bits of mantissa.

Target1 and Target2:
These glyphs generate arrays that differ by one in length compared to their Khoros 1.0 counterparts. For example, for the same parameter settings, the Khoros 1.0 version generates a 740x18x1x1x14 data cube whereas the Khoros 2.1 version generates a 741x18x14x1x1 data cube. Also note, that the depth (3rd) and elements (5th) dimensions are interchanged. Although the data is the same physical order in memory, its depth and elements dimension labels are interchanged. The rcheck script, mentioned in an earlier section, which is used to compare data files, is able to interchange the labels before comparisons are made. However, the rcheck script is not able to compare files that have different total sizes. Thus, the 740 vs 741 difference posed a problem.

Comparisons were further complicated by the fact that the random number generator seed values could not be specified in the Khoros 1.0 version.

The solution was to measure the statistics of the two files to determine if they are similar. A second script, rstats, was utilized to measure the means and standard deviations of the Khoros 1.0 and Khoros 2.1 version data files. The statistical comparison indicated that the two files were very similar. The means and standard deviations did not differ by more than 0.07 percent of full scale. Thus, the target glyphs are probably producing correct numbers, although the array lengths and sample spacings differ by one.

BNJam1 and BNJam2:
The Khoros 1.0 and Khoros 2.1 versions of these glyphs produced very different results. The differences were nearly an order of magnitude. A careful check of the jammer parameters did not reveal any differences in settings that might account for the results. It would appear that the Khoros 1.0 and Khoros 2.1 versions are implementing different algorithms, or there is a problem in one version or the other.

CltrMaps and CltrPlse:
These two glyphs are slightly different in the Khoros 1.0 versus Khoros 2.1 versions. Some of the calculations are partitioned differently between the two glyphs in the two versions. The following outputs were found to be statistically similar for the two versions:

```
- CltrMaps.DoppMap
- CltrMaps.EAngMap
- CltrMaps.TcvfMap
- CltrMaps.THgtMap
- CltrMaps.GAngMap
- CltrMaps.GCofMap

  The following outputs were found to be dissimilar indicating
that the algorithms are not the same in the two versions:

- CltrMaps.BackMap
- CltrPlse.Imap

RPESum:
  Since the RPESum inputs include the BNJam1 and BNJam2 glyph
outputs, their inclusion caused the Khoros 2.1 version of
RPESum outputs to differ subtantially from those of the
Khoros 1.0 version. The RPESum glyph was re-tested with the
BNJam1 and BNJam2 inputs disconnected. This modification did
not change the outcome of the experiment. The two versions
produced statistically dissimilar results.

RcvrNoise, etc.:
  All subsequent glyphs could not be tested since there was
no common data from previous glyphs to provide proper stimulus.
```

5.4 Execution Speed Tests

Both the Khoros 1.0 and Khoros 2.1 versions of the *convbf* and *jdo* workspaces were executed on a Fujitsu ULtraSparc clone machine running SunOS 5.5.1. The UltraSparc clone had 128MBytes of system memory. The machine was restricted to a single user during the benchmarks. However, the binaries were accessed via a NFS mount which caused some minor differences between runs.

The Khoros 1.0 version was utilized in the form it was shipped from Rome Labs and was not re-compiled locally. The Khoros 2.1 version was compiled on a local SparcStation using the /opt/SUNWSpro c compiler, and the binaries were made accessible to the Sparc clone upon which the benchmarks were executed.

5.4.1 Overall Speed

The *convbf* and *jdo* workspaces were executed in both the Khoros 1.0 and Khoros 2.1 environments and their overall execution times were recorded. The results are as follows:

Workspace	Version	Run Number	Execution Time
convbf	K1	1	2 minutes 34 seconds
convbf	K1	2	2 minutes 25 seconds
convbf	K1	3	2 minutes 26 seconds
convbf	K1	4	2 minutes 27 seconds
convbf	K1	5	2 minutes 27 seconds
average	2 minutes 27.8 seconds		

Workspace	Version	Run Number	Execution Time
convbf	K2	1	1 minute 16 seconds
convbf	K2	2	1 minute 16 seconds
convbf	K2	3	1 minute 13 seconds
convbf	K2	4	1 minute 14 seconds
convbf	K2	5	1 minute 13 seconds
average	1 minute 14.4 seconds		

Workspace	Version	Run Number	Execution Time
jdo	K1	1	4 minutes 41 seconds
jdo	K1	2	4 minutes 44 seconds
jdo	K1	3	4 minutes 43 seconds
jdo	K1	4	4 minutes 26 seconds
jdo	K1	5	4 minutes 52 seconds
average	4 minutes 41.2 seconds		

Workspace	Version	Run Number	Execution Time
jdo	K2	1	1 minute 36 seconds
jdo	K2	2	1 minute 27 seconds
jdo	K2	3	1 minute 23 seconds
jdo	K2	4	1 minute 32 seconds
jdo	K2	5	1 minute 32 seconds
average	1 minute 30.2 seconds		

Thus, the RLSTAP_K2 version of the *convbf* workspace executed nearly twice as fast than its RLSTAP_K1 counterpart, and the RLSTAP_K2 version of the *jdo* workspace executed approximately three times faster than its RLSTAP_K1 counterpart. Although the comparisons cannot be considered absolute, since the two versions were not compiled in the same manner, the results are indicative that the RLSTAP_K2 versions are more efficient than their RLSTAP_K1 counterparts.

5.4.2 Timings of Individual Routines

The execution speed of individual routines within the two workspaces was measured. Scripts were constructed from the Khoros 1.0 workspaces that utilized the Unix *timex* function to obtain measurements. The execution speed of individual routines in the Khoros 2.1 version was obtained utilizing the timing functions provided with the QSERVER logging mechanism. An attempt was made to disable the QSERVER (using QENABLE=0) and invoke the Unix *timex* function within a script, without success. Although the QSERVER mechanism reports the *user* and *sys* times, it does not report *real* time which includes time spent performing I/O. Thus, the *real* time measurement is unable for the Khoros 2.1 version.

The execution times (in seconds) for the convbf workspace were as follows:

Version	Function	user	sys	real
K1	WindowGen1	0.02	0.03	1.42
K1	WindowGen2	0.02	0.06	0.57
K1	WindowGen3	0.02	0.04	0.58
K1	WindowGen4	0.01	0.05	0.56
K1	PC	1.92	0.09	8.43

K1	Extract	0.28	0.09	6.45
K1	MotionComp	0.10	0.08	6.32
K1	MTI	0.14	0.14	5.95
K1	DopplerSubband	0.13	2.35	8.55
K1	ConvRule	0.05	0.17	5.60
K1	SteerVector	0.09	0.15	1.94
K1	FFT	0.04	0.06	0.89
K1	CnvBF	0.12	0.25	2.24
K1	Normalize	0.07	0.02	0.77
K1	Floor	0.07	0.04	1.12
K1	CA_CFAR	14.97	0.07	16.10
K1	GO_CFAR	16.14	0.07	17.33
K1	OS_CFAR	26.71	0.07	27.91
K1	TM_CFAR	28.15	0.06	29.26
K1	Detect1	0.14	0.07	1.53
K1	Detect2	0.14	0.07	1.51
K1	Detect3	0.15	0.05	1.54
K1	Detect4	0.13	0.06	2.24

Version	Function	user	sys
K2	WindowGen1	0.14	0.10
K2	WindowGen2	0.10	0.10
K2	WindowGen3	0.12	0.08
K2	WindowGen4	0.11	0.15
K2	PC	0.53	0.14
K2	Extract	0.27	0.12
K2	MotionComp	0.25	0.16
K2	MTI	0.41	0.10
K2	DopplerSubband	1.76	0.12
K2	ConvRule	0.17	0.10
K2	SteerVector	0.09	0.15
K2	FFT	0.04	0.06
K2	CnvBF	0.16	0.15
K2	Normalize	0.13	0.10
K2	Floor	0.17	0.12
K2	CA_CFAR	8.47	0.06
K2	GO_CFAR	9.24	0.19
K2	OS_CFAR	12.22	0.09
K2	TM_CFAR	12.81	0.07
K2	Detect1	0.26	0.13
K2	Detect2	0.25	0.11
K2	Detect3	0.26	0.05
K2	Detect4	0.22	0.19

The execution times (in minutes and seconds) for the jdo workspace were as follows:

Version	Function	user	sys	real
K1	WindowGen1	0.01	0.03	0.72
K1	WindowGen2	0.01	0.02	0.61
K1	WindowGen3	0.01	0.07	0.59
K1	WindowGen4	0.03	0.11	0.61
K1	PC	1.93	0.18	8.34

K1	Extract1	0.25	0.17	10.61
K1	MotionComp	0.09	0.10	6.44
K1	MTI	0.18	0.06	6.50
K1	JDORule	0.07	0.07	5.99
K1	SteerVector	0.12	0.12	2.47
K1	Covar	57.78	0.14	1:01.81
K1	DiagLoad	0.09	0.06	5.22
K1	InvCovar	56.71	5.07	1:05.48
K1	STAPWgts	1.47	0.09	4.07
K1	BeamForm	1.61	0.16	3.77
K1	Floor	0.03	0.07	1.14
K1	Extract2	0.12	0.07	2.16
K1	Extract3	0.11	0.06	1.92
K1	CA_CFAR	15.02	0.06	16.18
K1	GO_CFAR	16.14	0.09	17.58
K1	OS_CFAR	26.80	0.12	28.22
K1	TM_CFAR	28.20	0.07	30.19
K1	Detect1	0.15	0.02	1.66
K1	Detect2	0.13	0.08	2.04
K1	Detect3	0.14	0.06	2.25
K1	Detect4	0.14	0.06	1.66

Version	Function	user	sys
K2	WindowGen1	0.11	0.11
K2	WindowGen2	0.13	0.10
K2	WindowGen3	0.10	0.10
K2	WindowGen4	0.12	0.10
K2	PC	0.51	0.13
K2	Extract1	0.28	0.08
K2	MotionComp	0.29	0.09
K2	MTI	0.36	0.11
K2	JDORule	0.19	0.12
K2	SteerVector	0.23	0.16
K2	Covar	5.25	0.08
K2	DiagLoad	0.22	0.11
K2	InvCovar	5.59	3.35
K2	STAPWgts	0.40	0.19
K2	BeamForm	1.15	0.10
K2	Floor	0.21	0.08
K2	Extract2	0.20	0.16
K2	Extract3	0.23	0.13
K2	CA_CFAR	8.47	0.07
K2	GO_CFAR	9.37	0.07
K2	OS_CFAR	12.17	0.11
K2	TM_CFAR	12.79	0.08
K2	Detect1	0.31	0.07
K2	Detect2	0.26	0.07
K2	Detect3	0.25	0.11
K2	Detect4	0.24	0.05

5.5 Summary

Selected routines and workspaces from the RLSTAP_K2 environment were evaluated in terms of mathematical accuracy and throughput performance. Mathematical accuracy was determined by comparing the results of selected workspaces with that of their RLSTAP_K1 counterparts.

The *RSTERin* routine, used to import Matlab compatible files into the RLSTAP environment was tested and found to produce results equivalent to that of the RLSTAP_K1 version.

The *convbf* and *jdo* baseline workspaces were tested and found to produce results that were, in most cases, identical to those of their RLSTAP_K1 counterparts, and those routines that produced different results, differed only by one count in 24 bits. (1 part in 2 raised to the 24th power).

The *phymod* workspace functionality was incomplete in the current version of RLSTAP_K2. The most notable differences were in the Jammer model glyphs.

The execution speeds of the *convbf* and *jdo* workspaces were measured and compared with their RLSTAP_K1 counterparts. The new RLSTAP_K2 routines were 2 to 3 times faster than their RLSTAP_K1 counterparts.

6 Appendices

6.1 Appendix A - Parameter Settings for the Conventional Beamformer

The parameter settings for the conventional beamformer workspace are as follows.

```
PMSetup:
  Experiment Name:  RLSTAP ADT DEMONSTRATION
  CPI number:       1

  Radar Operational Range:
    Min Operational Range (km)   6.45
    Max Operational Range (km)  110.0

  Azimuth Simulation Parameters:
    Coordinate System   [ACS]
    Azimuth Start Angle (deg)  -180.0
    Azimuth Stop Angle  (deg)   180.0
    No. of Azimuth Cells        256

  Doppler parameters:
    No. of freq bins over the PRF    256

PMEnv:
  Model Initialization parameters:
    Specify Map or Simulated Clutter Type:
      [ON] Site Specific Clutter

    DMA/USGS Input File   ./Maps2/wsmr.cvr

    [OFF] Simulated Homogeneous Clutter
        [ON] Agricultural

    Curved Earth multiplier k:   1.3333

RcvPlat:
  Platform Latitude:
    Deg  33.7514   Min:   0.0     Sec   0.0

  Platform Longitude:
    Deg  -106.3720  Min:  0.0     Sec:  0.0

  Platform Altitude (km)         0.0384   [AGL]
  Platform Heading (deg GSCS)  270.0
  Platform Speed (m/sec)       172.0
  Platform Roll Angle  (Deg)   0.0
  Platform Pitch Angle (Deg)   0.0
  Platform Yaw Angle (deg)     0.0

  Receiver Phase Center Displacement:
     X  0.0     Y   0.0      Z  0.0
```

```
  Receive Data Sampling Freq:  (MHz)  1.0

AntPatCos:
  Pedestal level:  v_ped (dB)   -100
  Angle scale factor: k          1.0
  Exponent: x                    1.0
  No. of Spatial Pattern Samples 256

  Spatial Pattern Angles:
    Start (deg ACS)  -180.0
    Stop  (deg ACS)   180.0

WindowGen1:
  Number of points to generate     24
  Window Function Type  [Dolph Chebyshev]
    Peak Sidelobe Level (dB)  32.5

WindowGen2:
  Number of points to generate     14
  Window Function Type  [Rectangular]
    Start Index  1
    Stop Index  14

WindowGen3:
  Number of points to generate     14
  Window Function Type   [Rectangular]
    Start Index   1
    Stop Index   14

PMAntPatWin1:
  Element Pattern Type (Voltage):
    [Isotropic]

  Shape Factors:
    Pedestal Level:  v_ped (dB)  -100
    Angle Scale factor: k         1.0
    Exponent: x                   1.0

    Steering Angle (deg ACS)      0.0
    Number of elements            24
    Spacing / Wavelength          0.5
    Amplitude Error RMS (dB)      0.00000000
    Phase Error RMS  (dB)         0.00000000
    No. of spatial pattern samples 256

  Spatial Pattern Angles:
    Start  (deg ACS)   -90.0
    Stop   (deg ACS)    90.0

PMAntPatWin2:
  Element Pattern Type (Voltage):
```

```
    [Isotropic]

  Shape Factors:
    Pedestal Level:  v_ped (dB)  -100
    Angle Scale factor: k         1.0
    Exponent: x                   1.0

    Steering Angle (deg ACS)      0.0
    Number of elements            14
    Spacing / Wavelength          0.5
    Amplitude Error RMS (dB)      0.00000000
    Phase Error RMS  (dB)         0.00000000
    No. of spatial pattern samples 256

  Spatial Pattern Angles:
    Start  (deg ACS)  -180.0
    Stop   (deg ACS)   180.0

PMAntPatWin3:
  Element Pattern Type (Voltage):
    [Isotropic]

  Shape Factors:
    Pedestal Level:  v_ped (dB)  -100
    Angle Scale factor: k         1.0
    Exponent: x                   1.0

    Steering Angle (deg ACS)      0.0
    Number of elements            24
    Spacing / Wavelength          0.5
    Amplitude Error RMS (dB)      0.00000000
    Phase Error RMS  (dB)         0.00000000
    No. of spatial pattern samples 256

  Spatial Pattern Angles:
    Start  (deg ACS)   -90.0
    Stop   (deg ACS)    90.0

Normalize1:
  <no parameters>

Normalize2:
  <no parameters>

Normalize3:
  <no parameters>

RcvAnt:
  Receive Azimuth Parameters:
    Mechanical boresight Angle (deg PCS)  90.0
    Number of Azimuth Channels            14
```

```
    Element Spacing / Wavelength Ratio    0.5

  Receive Elevation Parameters:
    Mechanical Boresight Angle (deg PCS)  0.0
    Number of Elevation Channels          1
    Element Spacing / Wavelength Ratio    0.5

    Angular Rotation Rate  (RPM)          0.0
    Peak Aperture Gain (dB)              17.5
    Aperture Efficiency                  1.000000000
    X Aperture displacement (M)           0.0
    Y Aperture displacement (M)           0.0
    Z Aperture displacement (M)           0.0

XmtPlat:
  Reference to Receive Platform:
    Range (km)                            0.0
    Azimuth (deg GSCS)                    0.0
    Platform Altitude (km)                0.0384  [AGL]
    Heading (Deg GSCS)                  270.0
    Speed (meters/Sec)                  172.5
    Roll Angle (Deg.)                     0.0
    Pitch Angle (Deg.)                    0.0
    Yaw Angle (Deg.)                      0.0

  Transmitter Displacement:
    X:  0        Y:  0          Z:  0

XmtAnt:
  Transmit Mechanical Boresight Angle (Deg PCS):
    Azimuth:   90.0
    Elevation:  0.0

  Peak Aperture Gain (dB)            29.0
  Aperture Efficiency                 1.0000000

  Aperture Phase Center Location

    DX Displacement (M)   0.0
    DY Displacement (M)   0.0
    DZ Displacement (M)   0.0

XmtWave:
  Transmit Pulse Modulation Type:  [LFM]
     Chirp direction [up]

  General Parameters:
    Peak Transmit Power (KW)   86.3233
    Center Frequency (GHZ)      0.435
    Pulse Length (Usec)         50.0
    Pulse Repitition (Hz)     1000.0
```

```
    Number of pulses            18

  Polarization:      [HH Horizontal]

  Additional Parameters for:
    LFM Waveform:                          Barker Waveform:
    Transmit Banwidth (MHZ)    0.5
    Phase Droop (DEG)          0.0     No. of Layers (1 or 2)    1
    Phase Jitter (DEG)         0.0     Inner Barker Code, req'd  5
    Phase Offset (DEG)         0.0     Outer Barker Code, opt    5
    Waveform Delay (Usec)      0.0     (Legal Codes are: 2,3,4,5,7,11 and 13)

Target1:
  Target ID:  TARGET 1

  Target Parameters:
    Range to Target (KM)       60.0
    Azimuth Angle (Deg GSCS)   0.0
    Target Altitude (KM)       2.50731    [MSL]
    Target Heading (Deg. GSCS)  -65.6
    Target Speed (M/SEC)       200.0
    Radar Cross Section (dBsm)  5.0
    RCS Swerling Model          0
    RNG Seed                    0   (0 => process ID)

Target2:
  Target ID:  TARGET 2

  Target Parameters:
    Range to Target (KM)       67.5
    Azimuth Angle (Deg GSCS)   359.0
    Target Altitude (KM)       2.50731    [MSL]
    Target Heading (Deg. GSCS)  0.0
    Target Speed (M/SEC)       151.0
    Radar Cross Section (dBsm)  -10.0
    RCS Swerling Model          0
    RNG Seed                    0   (0 => process ID)

CltrMaps:
  Clutter Strength [normal (mean)]

  Season:   [summer]

  Terrain Spatial Backscatter Fluctuations  [exponential]

  Sea Spatial Backscatter Fluctuations  [exponential]

  Sea State Index       1
  RNG seed              0   (0 => use process ID)

CltrPlse:
```

```
  Control Parameters:
    Data Cube coordinates:
      Start Pulse Index   0
      Stop Pulse Index    0
      Start Channel Index 0
      Stop Channel Index  0   (0 = all)

  Terrain Temporal Backscatter Fluctuations:
   [non-fluctuating]
   Terrain Decorrelation time (1/e, msec)   500.0

  Sea Temporal Backscatter Fluctuations:
    [non-fluctuating]
    Sea Decorrelation Time (1/e, msec)    500.0

  RNG seed      0    (0 => process ID)

  selected pulse   1

BNJam1:
  Jammer ID:   JAMMER 1

  Control Parameters:
    Location Parameters: rel to receiver platform:
      Slant Range to Jammer (KM)      100.0
      Azimuth Angle (Deg GSCS)        310.0
      Jammer Altitude (KM)            1.0      [AGL]
      Jammer Heading  (Deg GSCS)      0.0
      Jammer Speed  (M/SEC)          100.0

  Operational Parameters:
    Radiated Power (ERP dBw)     30.0
    Transmit Frequency (GHz)     0.435
    Jammer Bandwidth (MHz)       100.0
    Jammer Period (MSEC)           1.0
    Jammer Duty Factor            0.25
    Jammer Delay (periods)        0.00000000

  RNG Seed   0    (0 => process ID)

BNJam2:
  Jammer ID:   JAMMER 2

  Control Parameters:
    Location Parameters: rel to receiver platform:
      Slant Range to Jammer (KM)      100.0
      Azimuth Angle (Deg GSCS)        320.0
      Jammer Altitude (KM)            1.0      [AGL]
      Jammer Heading  (Deg GSCS)      0.0
      Jammer Speed  (M/SEC)          100.0
```

```
  Operational Parameters:
    Radiated Power (ERP dBw)     30.0
    Transmit Frequency (GHz)     0.435
    Jammer Bandwidth (MHz)       100.0
    Jammer Period (MSEC)           1.0
    Jammer Duty Factor            0.25
    Jammer Delay (periods)        0.00000000

  RNG Seed   0     (0 => process ID)

RPESum:
  <no parameters>

RcvrNoise:
  Receiver Noise Parameters:
    Insertion Loss  (dB)                1.5
    Receiver Gain (dB)                196.0
    RMS gain variations (dB)            0.0
    Rcvr pre-IF Bandwidth (MHz)         0.8
    Rcvr Noise Figure (dB)              5.0
    Antenna view Temp (deg K)         200.0
    Lossy component Temp (deg K)      290.0
    Rcvr Reference Temp, To (deg K)   290.0

RcvrBP:
  Filter Bandwidth, 3dB  (MHz)     0.2
  Stop Band Bandwidth (MHz)        0.6
  No. of Freq. (FFT) points        0     (select default)

SignalLevels:
  <no parameters>

CNRCalc:
  Sys Noise Figure (dB)     5.0
```

6.2 Appendix B - Parameter Settings for Joint Domain Optimum

The parameter settings for Joint Domain Optimum workspace are as follows.

```
WindowGen1:
  Number of points to generate   100
  Window Function Type:   [Hamming]

PC:
  Weighting Domain   [TIME]
  Matched Filter Selection Mode  [Internally Generated]
  Internally Generated Weighting Function [WindowGen1.kdf]

  Matched Filter Pulse Modulation:  [LFM]
  Pulse Width  0.0        Bandwidth   0.0   (use transmit numbers)

  Chirp Mode  [UP]

Extract1:
  Starting Indices    1       x       1       x      1
  Ending Indices    403       x      16       x      1
  Index Increments    1       x       1       x      1

  Reshape Data:  [OFF]
  Array Size      1     x     1      x      1

Motion Comp:
  Motion Compensation Frequency:  [Use specified]   0.0

MTI:
  Number of pulses    3

JDORule:
  Range Processing Parameters:
    STAP Algorithm Configuration:  [Adaptive Array]

  Covariance Estimation (Secondary data set)
    [Use all bins]

  Beamforming (primary data set)
    [Use all bins]

Covar:
  <no params>

DiagLoad:
  Diagonal Load Level (dB)   -50.0
  With respect to:   [Diagonal Peak]

InvCovar:
  Inverse Algorithm:
    [CSVD Algorithm]
```

```
      [Use N Largest Eigenvalues]
        No. of eigenvalues   0

WindowGen2:
  Number of points to generate    14
  Window function type:  [Taylor]
    Peak Sidelobe Level (dB)  30.0
    NBAR                      5

WindowGen3:
  Number of points to generate    1
  Window function type:  [Rectangular]
    Start index   1
    Stop index    1

WindowGen4:
  Number of points to generate    14
  Window function type:  [Dolph Chebyshev]
    Peak Sidelobe level (dB)    80.0

SteerVector:
  [Doppler filter weights]  WindowGen4

  Steer Vec Az Angle (deg)  0.0
  Steer Vec El Angle (deg)  0.0

STAPWgts:
  RPE Data:  JDORule.kdf
  InverseCovariance:  InvCovar.kdf
  SteeringVector:     SteerVector.kdf

  Weight Scaling Option:  [Min Variance]

BeamForm:
  <no parameters>

Floor:
  Specification of Floor Value:
    [limit to dB, preserve phase]
    dB value   70.0

SignalLevels:
  <no parameters>

Extract2:
  Starting Indices     1       x      13       x      1
  Ending Indices     403       x      13       x      1
  Index Increments     1       x       0       x      0

  Reshape Data:   [OFF]
```

```
  Array size:     1     x      1      x     1

Extract3:
  Starting Indices     1       x      10       x      1
  Ending Indices     403       x      10       x      1
  Index Increments     1       x       0       x      0

  Reshape Data:   [OFF]

  Array size:     1     x      1      x     1

CA_CFAR:
  CFAR Parameters:
    Range Dimension:
      No. Guard Bins  (each side)   5
      No. Window Bins (each side)  20

    Doppler Dimension:
      No. Guard Bins   (each side)  3
      No. Window Bins  (each side)  5

  Wrap Option:
    [1-Doppler Wrap]

GO_CFAR:
  CFAR Parameters:
    Range Dimension:
      No. Guard Bins  (each side)   5
      No. Window Bins (each side)  20

    Doppler Dimension:
      No. Guard Bins   (each side)  3
      No. Window Bins  (each side)  5

  Wrap Option:
    [1-Doppler Wrap]

OS_CFAR:
  CFAR Parameters:
    Range Dimension:
      No. Guard Bins  (each side)   5
      No. Window Bins (each side)  20

    Doppler Dimension:
      No. Guard Bins   (each side)  3
      No. Window Bins  (each side)  5

  Wrap Option:
    [1-Doppler Wrap]

  Order Statistic:    3
```

```
TM_CFAR:
  CFAR Parameters:
    Range Dimension:
      No. Guard Bins  (each side)   5
      No. Window Bins (each side)  20

    Doppler Dimension:
      No. Guard Bins   (each side)  3
      No. Window Bins  (each side)  5

  Wrap Option:
    [1-Doppler Wrap]

  Number of Cells Excised:
    No. of Smallest Valued Cells   0
    No. of Largest Valued Cells    3

Detect1:
  Threshold Multiplier Specification:
    [Analytical]
      Pfa:  0.00001
      K:    0.0

Detect2:
  Threshold Multiplier Specification:
    [Analytical]
      Pfa:  0.00001
      K:    0.0

Detect3:
  Threshold Multiplier Specification:
    [Analytical]
      Pfa:  0.00001
      K:    0.0

Detect4:
  Threshold Multiplier Specification:
    [Analytical]
      Pfa:  0.00001
      K:    0.0
```

6.3 Appendix C - Parameter Settings for the Physical Model

The parameter settings for the physical model workspace are as follows.

```
PMSetup:
  Experiment Name:  RLSTAP ADT DEMONSTRATION
  CPI number:       1

  Radar Operational Range:
    Min Operational Range (km)   6.45
    Max Operational Range (km)  110.0

  Azimuth Simulation Parameters:
    Coordinate System    [ACS]
    Azimuth Start Angle (deg)  -180.0
    Azimuth Stop Angle  (deg)   180.0
    No. of Azimuth Cells        256

  Doppler parameters:
    No. of freq bins over the PRF    256

PMEnv:
  Model Initialization parameters:
    Specify Map or Simulated Clutter Type:
      [ON] Site Specific Clutter

    DMA/USGS Input File    ./Maps2/wsmr.cvr

    [OFF] Simulated Homogeneous Clutter
        [ON] Agricultural

    Curved Earth multiplier k:   1.3333

RcvPlat:
  Platform Latitude:
    Deg  33.7514   Min:   0.0     Sec   0.0

  Platform Longitude:
    Deg  -106.3720  Min:  0.0     Sec:  0.0

  Platform Altitude (km)         0.0384   [AGL]
  Platform Heading (deg GSCS)  270.0
  Platform Speed (m/sec)       172.0
  Platform Roll Angle  (Deg)   0.0
  Platform Pitch Angle (Deg)   0.0
  Platform Yaw Angle (deg)     0.0

  Receiver Phase Center Displacement:
     X  0.0     Y   0.0      Z  0.0

  Receive Data Sampling Freq:  (MHz)  1.0
```

```
AntPatCos:
  Pedestal level:  v_ped (dB)   -100
  Angle scale factor: k          1.0
  Exponent: x                    1.0
  No. of Spatial Pattern Samples 256

  Spatial Pattern Angles:
    Start (deg ACS)  -180.0
    Stop  (deg ACS)   180.0

WindowGen1:
  Number of points to generate     24
  Window Function Type  [Dolph Chebyshev]
    Peak Sidelobe Level (dB)  32.5

WindowGen2:
  Number of points to generate     14
  Window Function Type  [Rectangular]
    Start Index  1
    Stop Index  14

WindowGen3:
  Number of points to generate     14
  Window Function Type   [Rectangular]
    Start Index   1
    Stop Index   14

PMAntPatWin1:
  Element Pattern Type (Voltage):
    [Isotropic]

  Shape Factors:
    Pedestal Level:  v_ped (dB)  -100
    Angle Scale factor: k         1.0
    Exponent: x                   1.0

    Steering Angle (deg ACS)      0.0
    Number of elements            24
    Spacing / Wavelength          0.5
    Amplitude Error RMS (dB)      0.00000000
    Phase Error RMS  (dB)         0.00000000
    No. of spatial pattern samples 256

  Spatial Pattern Angles:
    Start  (deg ACS)   -90.0
    Stop   (deg ACS)    90.0

PMAntPatWin2:
  Element Pattern Type (Voltage):
    [Isotropic]
```

```
  Shape Factors:
    Pedestal Level:  v_ped (dB)  -100
    Angle Scale factor: k         1.0
    Exponent: x                   1.0

    Steering Angle (deg ACS)      0.0
    Number of elements            14
    Spacing / Wavelength          0.5
    Amplitude Error RMS (dB)      0.00000000
    Phase Error RMS  (dB)         0.00000000
    No. of spatial pattern samples 256

  Spatial Pattern Angles:
    Start  (deg ACS)  -180.0
    Stop   (deg ACS)   180.0

PMAntPatWin3:
  Element Pattern Type (Voltage):
    [Isotropic]

  Shape Factors:
    Pedestal Level:  v_ped (dB)  -100
    Angle Scale factor: k         1.0
    Exponent: x                   1.0

    Steering Angle (deg ACS)      0.0
    Number of elements            24
    Spacing / Wavelength          0.5
    Amplitude Error RMS (dB)      0.00000000
    Phase Error RMS  (dB)         0.00000000
    No. of spatial pattern samples 256

  Spatial Pattern Angles:
    Start  (deg ACS)   -90.0
    Stop   (deg ACS)    90.0

Normalize1:
  <no parameters>

Normalize2:
  <no parameters>

Normalize3:
  <no parameters>

RcvAnt:
  Receive Azimuth Parameters:
    Mechanical boresight Angle (deg PCS)  90.0
    Number of Azimuth Channels            14
    Element Spacing / Wavelength Ratio    0.5
```

```
  Receive Elevation Parameters:
    Mechanical Boresight Angle (deg PCS)  0.0
    Number of Elevation Channels          1
    Element Spacing / Wavelength Ratio    0.5

    Angular Rotation Rate  (RPM)          0.0
    Peak Aperture Gain (dB)              17.5
    Aperture Efficiency                  1.000000000
    X Aperture displacement (M)           0.0
    Y Aperture displacement (M)           0.0
    Z Aperture displacement (M)           0.0

XmtPlat:
  Reference to Receive Platform:
    Range (km)                            0.0
    Azimuth (deg GSCS)                    0.0
    Platform Altitude (km)                0.0384  [AGL]
    Heading (Deg GSCS)                  270.0
    Speed (meters/Sec)                  172.5
    Roll Angle (Deg.)                     0.0
    Pitch Angle (Deg.)                    0.0
    Yaw Angle (Deg.)                      0.0

  Transmitter Displacement:
    X:  0        Y:  0          Z:  0

XmtAnt:
  Transmit Mechanical Boresight Angle (Deg PCS):
    Azimuth:   90.0
    Elevation:  0.0

  Peak Aperture Gain (dB)            29.0
  Aperture Efficiency                 1.0000000

  Aperture Phase Center Location

    DX Displacement (M)   0.0
    DY Displacement (M)   0.0
    DZ Displacement (M)   0.0

XmtWave:
  Transmit Pulse Modulation Type:  [LFM]
     Chirp direction [up]

  General Parameters:
    Peak Transmit Power (KW)   86.3233
    Center Frequency (GHZ)      0.435
    Pulse Length (Usec)         50.0
    Pulse Repitition (Hz)     1000.0
    Number of pulses            18
```

```
  Polarization:      [HH Horizontal]

  Additional Parameters for:
    LFM Waveform:                          Barker Waveform:
    Transmit Banwidth (MHZ)    0.5
    Phase Droop (DEG)          0.0     No. of Layers (1 or 2)    1
    Phase Jitter (DEG)         0.0     Inner Barker Code, req'd  5
    Phase Offset (DEG)         0.0     Outer Barker Code, opt    5
    Waveform Delay (Usec)      0.0     (Legal Codes are: 2,3,4,5,7,11 and 13)

Target1:
  Target ID:  TARGET 1

  Target Parameters:
    Range to Target (KM)       60.0
    Azimuth Angle (Deg GSCS)   0.0
    Target Altitude (KM)       2.50731    [MSL]
    Target Heading (Deg. GSCS)  -65.6
    Target Speed (M/SEC)       200.0
    Radar Cross Section (dBsm)  5.0
    RCS Swerling Model          0
    RNG Seed                    0   (0 => process ID)

Target2:
  Target ID:  TARGET 2

  Target Parameters:
    Range to Target (KM)       67.5
    Azimuth Angle (Deg GSCS)   359.0
    Target Altitude (KM)       2.50731    [MSL]
    Target Heading (Deg. GSCS)  0.0
    Target Speed (M/SEC)       151.0
    Radar Cross Section (dBsm)  -10.0
    RCS Swerling Model          0
    RNG Seed                    0   (0 => process ID)

CltrMaps:
  Clutter Strength [normal (mean)]

  Season:   [summer]

  Terrain Spatial Backscatter Fluctuations  [exponential]

  Sea Spatial Backscatter Fluctuations  [exponential]

  Sea State Index       1
  RNG seed              0   (0 => use process ID)

CltrPlse:
  Control Parameters:
    Data Cube coordinates:
```

```
      Start Pulse Index   0
      Stop Pulse Index    0
      Start Channel Index 0
      Stop Channel Index  0    (0 = all)

  Terrain Temporal Backscatter Fluctuations:
   [non-fluctuating]
   Terrain Decorrelation time (1/e, msec)    500.0

  Sea Temporal Backscatter Fluctuations:
    [non-fluctuating]
    Sea Decorrelation Time (1/e, msec)     500.0

  RNG seed       0     (0 => process ID)

  selected pulse    1

BNJam1:
  Jammer ID:    JAMMER 1

  Control Parameters:
    Location Parameters: rel to receiver platform:
      Slant Range to Jammer (KM)      100
```

13. Enabling Online Help and Manual Pages for Khoros Pro 2.2

Paul DeLauretis
Maui Community College (MCC)
D.J. Fabozzi
Maui High Performance Computing Center (MHPCC)

21 August, 1998

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

ENABLING ONLINE HELP AND MANUAL PAGES FOR KHOROS PRO 2.2

Paul De Lauretis
Maui Community College
edoc@maui.net

D.J. Fabozzi
Maui High Performance Computing Center
fabozzi@mhpcc.edu

August 21, 1998

ABSTRACT

This document describes the procedures to build and install the Khoros Pro 2.2 manual package on the Maui High Performance Computing Center (MHPCC) AIX version 4.2 system. This project was born through the discovery that during the testing of the Rome Laboratory Space-Time Adaptive Processing High Performance Computing (RLSTAP_HPC) utilities, though the utilities functioned properly, the manual pages did not. This is function of the parent environment, Khoros2.2, not being supported for the IBM AIX system. As a result, the installation of the KhorosPro 2.2 manual page facility is nontrivial and requires many 3rd party products including GNU's `groff-1.11`, `geqn`, and `gtbl` text formatting utilities. Because of the extensive configuration involved in this process, this document describes the details of the process and utilities required for the generation of Khoros2.2 manual capability. Following this report are tables of related paths, commands, and locations of required 3rd party tools.

INTRODUCTION

The use of Khoros manual pages are necessary because of the significant quantity of information accompany its 200+ toolboxes. Unfortunately, Khoros is not presently supported on the IBM AIX platform so each AIX distribution requires a complete build from source code. Though the build of Khoros is somewhat straightforward, the build of the Khoros documentation or "manual" facility involves numerous additional 3rd party utilities. This document steps through the build and installation of those utilities and as well as provides additional information to assist other IBM and non-supported platform developers who wish to build the Khoros Pro 2.2 manual facility.

Many utilities are necessary to execute Khoros Pro 2.2 manual pages that are not included with the Khoros distribution. These include: sed-2.05.tar.gz, `autoconf-2.12.tar.gz`, `m4-1.4.tar.gz`, `automake-1.3.tar.gz`, `make-3.76.1.tar.gz`, `bison-1.25.tar.gz`, `gawk-3.0.3.tar.gz`, `groff-1.11a.tar.gz`. The process of building manual page capability involved installing and building these utilities in the listed order. Following this was the task of generating the binary description files for typewriter, or ascii type devices via the `makedev` and move the generated device distribution to a location pointed to by the BOOTSTRAP variable. This report discusses these configuration details in building these utilities for the KhorosPro 2.2 manual utility.

BACKGROUND

KhorosPro 2.2 is the flagship product of Khoral Reasearch Inc. of Albuquerque, NM. Khoros includes a toolbox of over 200 mathematical and data processing routines as well as a visual programming environment "Cantata". The Khoros utility which processes manual pages for display is entitled "kman" which in turn requires the GNU product `groff`.

Text formatting

A brief review of text formatting will introduce manual page generation. As mentioned, the Khoros man pages execute via the "kman" utility which in turn invokes `groff`. The formatter `groff` takes input files such as shown in figure 1, below.

```
TH COFFEE 1 "1 July 98"
.SH NAME
coffee \- Control remote coffee machine
.SH SYNOPSIS
\fBcoffee\fP [ -h | -b ][ -t \fItype\fP] \fIamount\fP
.SH DESCRIPTION
\fIcoffee\fP queues a request to the remote coffee machine at the device \fB/dev/cf0\fR.
The required \fIamount\fP arguments specifies the number of cups, generally between 0
and 15 on ISO standard coffee machines.
.SS Options
.TP
\fB-b\fP
Burn coffee. Especially useful when executing \fIcoffee\fP on behalf of your boss.
.TP
\fB-t \fItype\fP
Specify the type of coffee to brew, where \fItype\fP is one of \fBcolombian\fP,
\fBregular\fP, or \fBdecaf\fP.
.SH FILES
.TP
\fI/dev/cf0\fR
The remote coffee machine device
.SH "SEE ALSO"
milk(5), sugar(5)
.SH BUGS
May require human intervention if coffee supply is exhausted.
```

Figure 1

When processed with the following command,

```
groff -Tascii -man coffee.man
```

produces the output as shown in Figure 2.

COFFEE(1) (1 July 98) COFFEE(1)

NAME

coffee - Control remote coffee machine

SYNOPSIS

coffee [-h | -b] [-t type] amount

DESCRIPTION

coffee queues a request to the remote coffee machine at the device /dev/cf0. The required amount argument specifies the number of cups, generally between 0 and 15 on ISO standard coffee machines.

Options

-h Brew hot coffee. Cold is the default.

-b Burn coffee. Especially useful when executing coffee on behalf of your boss.

-t type

Specify the type of coffee to brew, where type is one of colombian, regular, or decaf.

FILES

/dev/cf0

The remote coffee machine device

SEE ALSO

milk(5), sugar(5)

BUGS

May require human intervention if coffee supply is exhausted.

Figure 2

Utilities

As mentioned, the installation of groff requires the operation of numerous utilities which are available independent of KhorosPro. Though some were already installed on the MHPCC system, many of these utilities required specific version and additional attachments. These additional utilities are summarized below:

sed-2.05

sed is a common tool used for stream-text editing, having ed-like syntax. Sed can also operate on files using a script in edit lines according to commands and.

autoconf-2.12

Autoconf is an extensible package of m4 macros that produce shell scripts to automatically configure software source code packages. These scripts can adapt the packages to many kinds of UNIX-like systems without manual user intervention. It creates a configuration script for a package from a template file that lists the operating system features that the package can use in the form of m4 macro calls.

m4-1.4

GNU's m4 is an implementation of the traditional UNIX macro processor. It is mostly SVR4 compatible, although it has some extensions to BSD unix (for example, handling more than nine positional parameters to macros). m4 has built-in functions for including files, running shell commands, doing arithmetic, etc. autoconf requires GNU's m4 for generating configure scripts, but not for running them.

automake-1.3

automake is a Makefile generator. It was inspired by the 4.4BSD make and include files, but aims to be portable and to conform to the GNU standards for makefile variables and targets. automake is a Perl script and requires Makefile.am inputfiles and generates Makefile.in output files to be used with autoconf. The automake package also includes the aclocal program which is required to make to execute properly.

make-3.76.1

The make utility determines performs robust compilation of large software packages. GNU's make was utilized because IBM's make did not recognize the "FORCE" rule. Additional details follow in the "Approach" section of this document.

bison-1.25

Bison is a parser generator for converting a grammatical specification of a language into a parser that will parse statements in the language. A grammar is written which specifies the syntax of the language. Bison can then be used to warn of errors and ambiguities in the grammar. It should be upwardly compatible with input files designed for yacc. It supports both traditional single-letter options and mnemonic long option names.

gawk-3.0.3

gawk is the GNU implementation of the AWK programming language. The AWK language is built upon C syntax and includes the regular expression search facilities of grep and add in the advanced string and array handling features that are missing from C. Gawk also provides more recent Bell Labs AWK extensions, and some GNU-specific extensions.

groff-1.11

groff is GNU's implementation of nroff and troff document formatting system. It runs gtroff, GNU's version of troff, and all preprocessors and postprocessors in order and with the appropriate options. It includes several extended features and drivers for a number of printing devices. Groff is capable of producing documents, articles, and formatting UNIX manual pages. The groff package also contains the required geqn and gtbl formatting tools.

The interdependencies of the above tools accentuated this effort. For instance GNU's autoconf is to be used to run the configure process for most GNU utilities. Further, autoconf in turn requires m4 and

automake to build correctly. In addition, GNU's make requires automake and autoconf and finally groff requires make as well as bison, sed, and gawk. The remainder of this report discusses these dependencies in the context of the build and installation process on the IBM SP environment.

APPROACH

The installation of GNU's groff-1.11 was undertaken through the following steps. The approach was straightforward: obtain and install all the necessary third party products in a user's directory rather than the system "root" locations.

installbsd

As on of the objectives of this exercise was to perform the installation of these utilities without system or "root" privelege, the installbsd utility was employed to installs files to specified locations by adjusting the ownership option of the Makefile's INSTALL macro to show group ownership. A unique property of the installbsd command is its ability to remove the binary destination file. This removal feature makes installbsd ideal for use in Makefiles as the continual rebuilding of dependency files requires the most recently built version of a binary executable file to be present.

The syntax of the installbsd command is as follows:

```
<path >/installbsd [ -c ] [-g group ] [ -m Mode ] [ -o owner
] [ -s ] BinaryFile Destination
```

And the value assigned to the Makefile's macro was as follows:

INSTALL=/usr/bin/installbsd -c -g <group>

Here the group ownership of the target directory was used as <group>.

sed

The GNU version of sed was required over the AIX version due to a specific command reference. The AIX version sed was unable to recognize "@g@" in the neqn.sh file with the value of g given in the Makefile. As a result sed-2.05 was downloaded and used in this exercise.

make

GNU's make is needed because of the use of "phony targets" which are not supported by the AIX version of make. By using the FORCE identifier rather than the GNU specific "PHONY" identifier, a rebuild is forced. A "phony target" is one that is not really the name of a file but rather a name for some commands to be executed when you make an explicit request. There are two reasons to use a phony target: to avoid a conflict with a file of the same name, and to improve performance.

For instance, if you write a rule whose commands will not create the target file, the commands will be executed every time the target comes up for remaking. Here is an example:

```
clean:
        rm *.o temp
```

Because the rm command does not create a file named `clean', probably no such file will ever exist. Therefore, the rm command will be executed every time you say `make clean'. The phony target will cease to work if anything ever does create a file named `clean' in this directory. Since it has no dependencies, the file `clean' would inevitably be considered up to date, and its commands would not be executed. To avoid this problem, you can explicitly declare the target to be phony, using the special target PHONY as follows:

```
.PHONY : clean
clean:
        rm *.o temp
```

Once this is done, `make clean' will run the commands regardless of whether there is a file named `clean'. Utilizing "FORCE" yields equivalent results:

```
clean: FORCE
        rm $(objects)
FORCE:
```

Here the target `FORCE' satisfies the special conditions so the target `clean' that depends on it is forced to run its commands. There is nothing special about the name `FORCE', but that is one name commonly used this way. Using `PHONY' is more explicit and more efficient but other versions of make do not support `PHONY'.

If a rule has no dependencies or commands, and the target of the rule is a nonexistent file, then make imagines this target to have been updated whenever its rule is run. This implies that all targets depending on this one will always have their commands run.

Since it knows that phony targets do not name actual files that could be remade from other files, make skips the implicit rule search for phony targets (see section Using Implicit Rules). This is why declaring a target phony is good for performance, even if you are not worried about the actual file existing. Thus, you first write the line that states that clean is a phony target, then you write the rule, like this:

```
.PHONY: clean
```

A phony target should not be a dependency of a real target file. If it is, its commands are run every time make goes to update that file. As long as a phony target is never a dependency of a real target, the phony target commands will be executed only when the phony target is a specified goal.

After GNU `make` was properly installed, new makefiles were created for the utilities `bison`, `gawk`, and `groff-1.11`.

groff

The installation of `make` and the other utilities was followed by the build of `groff-1.11`. `Groff` was built with c rather than c++ because the egcs-1.0.3a package which includes support for most of the current c++ specification, including template and exception handling, was not resident on the system and difficult to externally obtain.

./configure --srcdir=/s/edocstud/groff-1.11 --prefix=/s/edocstud/groff-1.11 --exec-prefix=/s/edocstud

groff then built without error

DESC.out file

It was discovered next that even though groff was functional, Khoros' kman required an additional font configuration file which contains a brief description of the device which is needed for the use of a typewriter or ascii type device. As the file is not resident on the MHPCC's /usr/lib/font directory, makedev was executed and the DESC.out binary file and the four relevant font binary files were built according to the following command:

```
makedev ./DESC
```

ONLINE HELP

Following the completion of these steps, the execution of kman on the help files produced the manual page interface, as shown in Figure 3.

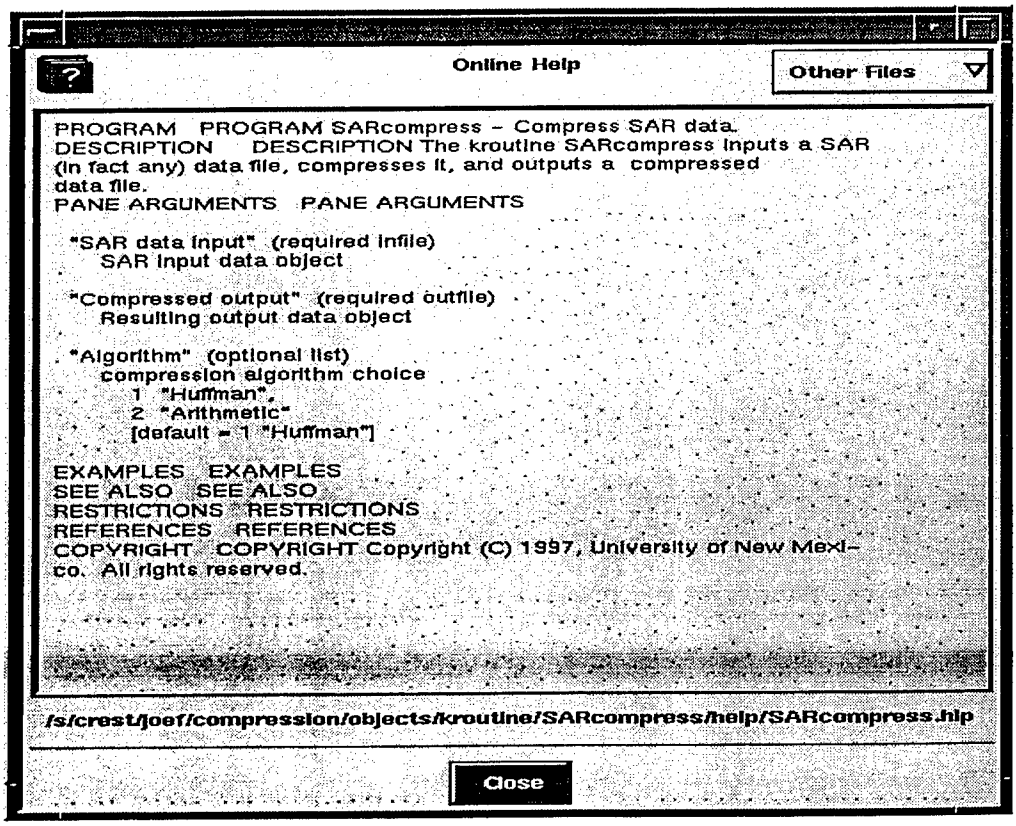


Figure 3

SUMMARY

Khoros manual pages are necessary because of amount of information accompanying the Cantata utility. Developers as well as users of Cantata require the help utility for both execution as well as development of new modules. As Khoros is not presently supported on the IBM AIX platform, each distribution to an AIX platform involves a compilation and build of the Khoros distribution. Though the build of Khoros system is straightforward the build of the Khoros manual facility is not. This document details the steps involved and the required external software to assist other IBM and non-supported platform developers who wish to work the Khoros Pro 2.2.

APPENDIX

TABLE OF RELATED PATHS

Item	Location
Khoros2.2 3rd Party Requirements	ftp.khoral.com/pub/khoros/Khoros2/release/ installguide.tar.gz
GNU Software	ftp.gnu.org/pub/gnu
Target Directory	/s/edocstud/
Tools (bin) Directory	/s/edocstud/bin
C Compiler Location	/usr/bin
C++ Compiler Location	/usr/nfs/bin
DESC File Location	/s/edocstud/groff-1.11/share/groff/font/devascii/
Khoros2.2	/usr/nfs/packages/khoros2.2

TABLE OF RELATED COMMANDS

Command	Implementation
configure	./configure –srcdir=/s/edocstud/<build dir> --prefix=/s/edocstud/<build dir> --exec-prefix=/s/edocstud
installbsd macro	INSTALL=/usr/bin/installbsd -c -g asp
CXX env. variable	export CXX=/usr/bin/cc
PATH env. variable	export PATH=/s/edocstud/bin:/usr/bin:/usr/nfs/bin:$PATH
groff test	groff -Tascii -man coffee.man
makedev	makedev /s/edocstud/groff-111/share/groff/font/devascii/DESC
symbolic link	ln -s /usr/nfs/packages/khoros2.2/bootstrap/repos /s/edocstud/groff-1.11/share/groff/font/repos
BOOTSTRAP env. variable	export BOOTSTRAP=/s/edocstud/groff-1.11/share/groff/font

TABLE OF TOOL LOCATIONS

- All tools are located at ftp.gnu.org

Utility	Author(s)	location
sed-2.05.	Free Software Foundation, Inc	/pub/gnu/sed2.05.tar.gz
autoconf-2.12	David MacKenzie djm@catapult.va.pubnix.com	/pub/gnu/autoconf-2.12.tar.gz
m4-1.4	Free Software Foundation, Inc.	/pub/gnu/m4-1.4.tar.gz
automake-1.3	David MacKenzie djm@catapult.va.pubnix.com	/pub/gnu/automake-1.3.tar.gz
make-3.76.1	Paul D. Smith <psmith@gnu.ai.mit.edu>.	/pub/gnu/make-3.76.1.tar.gz
groff-1.11	James Clark <jjc@jclark.com>	/pub/gnu/groff-1.11a.tar.gz
gawk-3.0.3	Free Software Foundation, Inc.	/pub/gnu/gawk-3.0.3.tar.gz
bison-1.25	Free Software Foundation, Inc.	/pub/gnu/bison-1.25.tar.gz

14. An Examination of Parallelism with MultiMATLAB

Matthew Green
University of Massachusetts, Amherst
D.J. Fabozzi
Maui High Performance Computing Center (MHPCC)

24 August, 1998

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

An Investigation of Parallelization of Code Written in MATLAB

D.J. Fabozzi II, Blaise Barney
Maui High Performance Computing Center
fabozzi@mhpcc.edu, blaise@mhpcc.edu

Peter Young
Environmental Research Institute of Michigan
young@erim-int.com

Paul De Lauretis, Melody Bohn
Maui Community College
edoc2@aloha.net, kas@maui.net

Mathew Green
University of Massachusetts
mgreen@cs.umass.edu

ABSTRACT

The MathWorks' MATLAB is a very popular tool among scientists and engineers due to its rich mathematical library set, flexibility, and ease of use. However, its interpretive operation is an obstacle to high performance on either embedded processors or Massively Parallel Processors. As an alternative to the manual translation of MATLAB syntax to a compilable language, there have been a number of utilities which perform language conversions, integrate parallel libraries into the MATLAB user code, or a combination of both. It was found that the degree of improvement achieved by each of these developments was dependent on many factors such as MATLAB code structure, communication interface, and execution environment. This report reviews the various approaches to improve MATLAB and further details the results of an on-site evaluation of three third-party utilities. In particular, the MathWorks MATLAB compiler [1], MultiMATLAB [2], and Real Time Express [3] utilities were evaluated at the Maui High Performance Computing Center to examine performance, applicability, availability, and development cost when utilized on a diverse set of test codes. Following the results of the on-site evaluation, recommendations are given for both user and utility developers to improve MATLAB computing in a high performance environment. This work was funded by the Defense Advanced Research Projects Agency/ Sensor Technology Office (DARPA/STO) under Air Force contract F30602-95-C-0117.

1 INTRODUCTION

MATLAB is a very popular tool among scientists and engineers due to its rich mathematical library set, flexibility, and ease of use. However, its interpretive operation can be an obstacle to achieving high performance on Massively Parallel Processors (MPP). As a result, there have been a number of utilities to assist users in improving MATLAB code performance on either single or distributed processors through either conversion to compiled languages, substituting message passing within the MATLAB user code or a combination of both. The degree of improvement achieved by each of these utilities, however, depends on many factors such as MATLAB source code structure, improvement goal, labor, MATLAB licenses, and available computing resources. This report surveys the related MATLAB-based computing techniques and further details the results of the on-site evaluation of three leading solutions: the MathWorks' MATLAB compiler, Cornell Theory Center's MultiMATLAB, and Integrated Sensors Incorporated Real Time Express. Each solution was evaluated against a diverse suite of codes to demonstrate various performance and integration details.

1.1 Survey Of Related Work

The results of our survey of found that the field of work of improving MATLAB performance can be classified into three categories:

1. Compiler approaches—utilities which convert MATLAB code to compiled machine language.
2. Interpretive approaches—utilities which allow MATLAB algorithms to be distributed and executed within the MATLAB environment on independent processors.
3. "Full Suite" approaches—utilities which convert MATLAB code to compiled code such as C/C++ and subsequently insert parallel constructs into the generated language.

These are summarized in Table 1:

Category	Title	Developer Info	Approach	Availability	Reference	Notes
1	FALCON	L. DeRose, D. Padua, CSRD, University of Illinois	MATLAB to Fortran 90 Translator	Authors	www.csrd.uiuc.edu/falcon/	reference
1	MATCOM	MathTools, Ltd.	MATLAB to C/C++	COTS	www.mathtools.com/	Version 2 tested at MHPCC
1	mcc	MathWorks, Inc	MATLAB to C/C++	COTS	www.mathworks.com/	Version 2.0.1 tested at MHPCC
2	Matpar	Paul L. Springer, Jet Propulsion Laboratory	Interpretive HPC	Author	www-hpc.jpl.nasa.gov/PS/MATPAR/index.html	reference
2	MultiMATLAB	Anne E. Trefethen, Cornell Theory Center	Interpretive HPC	Author	www.tc.cornell.edu/~anne/projects/MM.html	Tested at MHPCC
2	Parallel Toolbox (PT)	Pauca, Liu, et al. Wake Forest U.	Interpretive HPC	ftp://web.mthcsc.wfu.edu/pub/pt	www.mthcsc.wfu.edu/pt/pt.html	reference
3	RTExpress	Integrated Sensors Inc	Translation to C, MPI	COTS	http://www.rtexpress.com/	Tested at MHPCC

Table 1. Survey of Leading Approaches

1.2 Review Of Evaluated Parallel-MATLAB Utilities

A brief description of the MATLAB test codes will properly introduce the evaluation results. The test codes were selected based on size, complexity, and code structure. The first set, referred to as the "Falcon" codes[4], consisted of eleven independent algorithms of 100 lines or less containing data structures of 500 elements or less. The second set of MATLAB test codes[5] consisted of variations of FFT processing, nested loops, and matrix multiply operations. The third test code [6] consisted of a Space-Time Adaptive Processing (STAP) distribution consisting of 47 library routines containing on average 100 lines of code. Where applicable, each code suite was tested with each third-party utility.

The first utility examined was the MathWorks' MATLAB compiler, which converts MATLAB code to C/C++. The converter/compiler demonstrated varying success depending on code style, use of data types (i.e. INTEGER, REAL, COMPLEX, LOGICAL), rank (i.e. SCALAR, VECTOR, MATRIX) and shape of arrays (i.e. size of each dimension). We found, for example, the MATLAB compiler data type inference provided speed-up ranging from 1 to 581 on the Falcon codes, depending on the use of complex numbers, "for" loops, and matrices. However, the MATLAB compiler provided only a negligible speedup on the STAP codes because of the strong use of complex numbers and MATLAB library routines. These as well as other factors including dependency on matrix element access and MATLAB callback routines are discussed in this report.

The second utility, Cornell Theory Center's MultiMATLAB, allows users to integrate parallel MEX routines into serial MATLAB programs for multiprocessor execution.

Message Passing Interface (MPI)—based parallel routines are called from within the interpretive MATLAB environment to perform process control, message passing, data distribution, and arithmetic operations. Though MultiMATLAB allows for the quick development of parallel MATLAB software, the utility was found to improve algorithm performance only for data parallel models in which data transfer time is only a small fraction of the computation time. Our testing revealed that the matrix multiply, FFT, matrix inverse and eigenvalue test codes all achieved improvement, but the improvement exhibited strong dependence on the selection of block gathering method.

The most aggressive approach to integrating parallel constructs into MATLAB is achieved by Integrated Sensors' RTExpress™ (Real-Time Express). RTExpress performs a conversion of MATLAB source to C followed by the automatic integration of message passing constructs and parallel libraries. Testing of RTExpress found measurable improvement for vector operations, FFT's, and data intensive parallel operations but found that it required a certain amount of training and that it did not perform well on code containing "for" loops.

While no utility offers a universal solution to improving MATLAB performance, this report illustrates various factors which influence the performance of MATLAB code in a High Performance Computing (HPC) environment. This report concludes with recommendations for both third party utility developers and MATLAB users to improve the development of future High Performance Computing applications with MATLAB. Appendix A lists all of the relevant MATLAB source codes used in the testing.

2 COMPILER APPROACHES

The compiler approaches are those that translate the MATLAB syntax to a compilable language such as C, C++, or Fortran 90. We examined three projects, one from academia and two from commercial companies.

2.1 FALCON - A MATLAB to Fortran90 Translator

This project involved the use of a translator to convert MATLAB code to Fortran 90. This project built inference mechanisms to determine functions and variable characteristics from the MATLAB syntax. Both *static* and *dynamic* inference strategies were implemented

against a simplified branching model for variable identification properties including type (i.e. integer, real, complex, or logical), rank (i.e. scalar, vector, or matrix), and shape (i.e. size of each dimension). Through the use of both forward and backward propagation strategies, the project achieved results which were not only comparable to hand written Fortran 90 equivalent code but superior to results achieved through the use of the MATLAB compiler.

Some of FALCON's test parameters are explained. The FALCON project's static inference mechanism uses a Static Single Assignment (SSA) representation of MATLAB programs in which each variable is assigned a value by at most one date type. The intrinsic types of variables are determined individually by the interpreter from the values of the operands. For non-SSA test code which contained variables whose types which cannot be determined immediately, such as those near looping assignments, types are temporarily assigned and resolved through forward and backward analysis of the MATLAB code.

Similarly, rank information is inferred by the examination of the operands as well during the same compiler pass. Rank determination is very important because if all data types are converted to the default matrix type, the computational labor to process and manage matrix data increases significantly in comparison to other inference classes. Lastly, though shape is found similarly to rank information, it was found to be best inferred through both static and dynamic examinations. Shape information is influential in code efficiency because element-wise access to matrices results in the computational overhead of dynamic allocation.

The FALCON project instruments dynamic type and shape inference through the use of shadow variables and conditional statements that are inserted into the code under test during execution. Also implemented is the use of a dimension propagation algorithm in which variable index accesses are counted so as to inform the resulting Fortran version how to implement memory requests. In other words, it was found that the overhead due to MATLAB memory requests with each increase in a matrix's size could be minimized if the final size of the matrix was known early and allocated only once.

The FALCON project tested these algorithms against 10 MATLAB codes which perform varying computational actions, the results of which are summarized as follows. The conversion to a compiled language will yield little performance improvement if the MATLAB code uses mostly complex matrices, spends most of its time performing library calls and does not perform incremental array indexing. Codes converted to Fortran 90 which

utilize scalar variables will be dramatically improved through the use of type inference. The use of shape inference against these same codes, however, will only improve them proportional to the number of matrices used. Compiled codes that will experience the greatest speedup over their MATLAB counterparts are those which utilize non-complex data types and little use of library routines.

When compared to the Mathwork's MATLAB compiler, the FALCON translator demonstrated superior performance due to the preallocation of variables and the handling of small dimension matrix operations.

The disadvantages of the FALCON compiler include support and testing. As this work is not a commercial product, its support and availability are questionable. Also, it is yet to be determined how the compiler performs on MATLAB subroutines over 60 lines as well as on a suite of operational test code.

2.2 MathTools' MATCOM – A MATLAB to C/C++ Compiler

MATCOM is a MATLAB to C/C++ compiler, much like the MathWork's compiler. As with the MathWork's compiler, MATCOM can create MEX files as well as standalone applications. Regarding the C++ objects produced, MATCOM uses templates "T" and declares the base object "M" as an instance of the class "T". MATCOM does provide the global inference of a `float` data type and does allow global specification of types with variable names. As of this writing MATCOM contains many features the MathWork's compiler does not including graphical user interface (GUI) functions (Windows only), sparse matrix support, multidimensional imaging support, enhanced error reporting of error location, compilation of S-funcs for Simulink, m-files for RTWLoad/save of V5 mat-files format, and recursive function search. This product was briefly evaluated at the MHPCC but was found to have substandard error reporting. Another feature we found useful in MATCOM was the recursive subroutine search which MATLAB did not perform at the time of this writing. For the large STAP distributions the MATCOM compiler recursively searched all routines that were called by the original function, where the MathWork's compiler needed to be independently invoked for each function call.

2.3 MathWorks' MATLAB Compiler

Mathworks, Inc. provides a facility to convert MATLAB code to C language code in either a standalone or as a MATLAB executable callable from MATLAB. The MathWork's compiler contains similar inference mechanisms as MATCOM for imputing type information or matrix handling. The MathWork's compiler V1.2 offered two command line switches (–r and –i) which perform imputation of data types. The "–i" compiler option generates code that fixes array size, eliminates boundary checking and type checking. The "-r" switch on the other hand tells the compiler to convert all data types to real rather than complex. The MATLAB compiler also has the ability to control the type imputations by specifying particular MATLAB variables individually through pragmas, which impute on a file basis. The pragmas also include a %#ivdep switch which tells the compiler to ignore vector dependencies. The MathWork's compiler also contains three special functions—reallog, realpow, and realsqrt which are real-only versions of log, .^, and sqrt functions.

The MathWork's compiler contains a library of routines which can be manually substituted for MATLAB language routines to improve performance over callbacks to the MATLAB interpreter. However, these libraries were not investigated for this report.

Operation of the MathWork's compiler on small sets of code is straightforward but the conversion of a full suite of code of requires attention to each called subroutine and more operator intervention. Utilities were developed to facilitate the use of MATLAB "load," "eval," "save," and "input" commands which are not handled by the MathWork's compiler.

Table 2 lists the results of the conversion of the FALCON test suite with the MathWork's compiler. The performance of the test code after translation through the MATLAB compiler's in general agreed with both our expectations and with the FALCON report results.

Code		Problem Size	Interpreted	mcc	mcc -ri *	Speedup
AQ	Adaptive Quadrature using Simpson's rule	1x7	9.02	3.05		2.97
CG	Conjugate Gradient method	420x420	14.73	0.33		44.64
FD	Finite Difference Solution to Wave Equation	451x451	11.76	0.45	0.23	51.13
Di	Dirichlet Method for Laplace's Equation	40x40	17.45	0.82	0.03	581.67
Ga	Poisson equation solved using Galerkin method	40x40	12.55	0.82		15.3
EC	Euler method to compute orbit of a comet	6240 steps	6.83	2.48		2.75
RK	Runge-Kutta method	3200 steps	8.94	7.97		1.12
IC	Incomplete Cholesky matrix factorization	400x400	11.35	0.94	0.24	47.29
3D	Generation of 3D-surface	51x31x21	14.37	5.34		2.69

Table 2. "FALCON" Code Conversion Results

* - ri switches are not applicable for all test codes

In general all test codes yielded speedup through compilation and syntax assumptions made earlier were consistent. The conversion to a compiled language yielded little performance improvement if the MATLAB code used complex matrices, library calls and array indexing. Degrees of improvement are proportional to these primary dependencies.

3 INTERPRETIVE APPROACHES

3.1 Matpar: Parallel Extensions to MATLAB

Of the interpretive approaches which insert the parallel constructs directly in MATLAB syntax, Matpar[7] incorporates PVM message passing utilities for data movement. These routines, which provide access to Scalapack, PBLAS, BLAS, and BLACS libraries, are listed in Table 2.

COMMAND	ACTION
p_config()	Specify parallel computer and quality of nodes
p_add()	Matrix add
p_subtract()	Matrix subtract
p_bode()	Generate frequency response of a matrix
p_delete()	Delete a persistent matrix
p_eye()	Generate an identity matrix on the server
p_freqresp()	Generate frequency response of a matrix
p_inv()	Matrix inverse
p_lu()	LU factorization of a matrix
p_mult()	Matrix multiply
p_multtrans()	Matrix transpose multiply
p_persist()	Keep matrix on remote server
p_pinv()	Pseudoinverse
p_qr()	QR factorization
p_smult()	Scalar matrix multiply
p_solv()	Matrix division
p_svd()	Single value decomposition
p_trace()	Compute trace

Table 3. Available Matpar Commands

Matpar was developed and tested on a variety of architectures including HPExamplar, SunUltra SPARC, Intel Paragon, and Cray T3-D.

Though Matpar was not tested on site, the documentation indicates that the implementation does not require multiple MATLAB licenses for each distributed process. Matpar's limitations, however, are in its limited number of libraries implemented and limited distribution.

3.2 PT: Parallel Toolbox for MATLAB

Parallel Toolbox (PT)[8] allows distributed MATLAB programs to execute in a Single Program Multiple Data (SPMD) configuration. PT also uses PVM to implement a master-slave paradigm for concurrent program execution. Though similar in using PVM, PT contrasts with Matpar in that it provides both data management and process control. Table 4 lists some of the available PT commands.

COMMAND	ACTION
pt_addEngines()	add one or more compute engines
pt_barrier()	synchronize members of a group
pt_broadcast()	broadcast a matrix to a group
pt_cleanup()	cleanup after job failure
pt_delete()	Delete a compute engine from a group
pt_erReset()	Resets error log file
pt_exit()	exit a worker from the group
pt_getinst()	determine the instance # from a group
pt_getwid()	determine worker id
pt_getsize()	get the # of members of a group
pt_hosts()	find all hosts where engines are present
pt_joingroup()	Enroll a worker in a group
pt_kill()	kill PT tasks in PVM
pt_lvgroup()	remove worker from a group
pt_mywid()	get worker id
pt_send()	send a matrix to a worker
pt_recv()	receive a matrix from another worker
pt_shutdown()	shutdown engines

Table 4. Available Parallel Toolbox Commands

Though PT was not evaluated on-site, the documentation revealed that this implementation requires MATLAB licenses for each worker node. Lastly, like Matpar, PT lacks significant distribution.

3.3 MultiMATLAB: MATLAB on Multiple Processors

MultiMATLAB is similar to PT in that it provides data distribution library routines which are interleaved directly into the original MATLAB source code. MultiMATLAB was developed on an IBM SP system utilizing the P4 implementation of MPICH for the underlying communication. Table 5 lists the available MultiMATLAB commands.

Most commands can be run only on the master process (process 0).
*Commands marked by * can be run on the master process*
or on the remote processes (1:Nproc-1).

Starting and Stopping MultiMATLAB.

Start	- Initialize remote processes and begin MultiMATLAB session
Interrupt	- Interrupt MultiMATLAB processes during computation
Abort	- Abort MultiMATLAB session remaining in originally interactive session
Quit	- Terminate remote processes and end MultiMATLAB session

Process Arrangement and Identification.

*ID	- Task ID of a process
*Nproc	- Total number of MultiMATLAB processes active
Grid	- Arrange the processes in a grid
*Gridsize	- Dimensions of the grid of processes
*Coord	- Coordinates of a process in the grid

Running Commands on Multiple Processes

Eval	- Evaluate a command on one or more processes

Communication

*Send	- Send data from one process to another
*Recv	- Receive data sent from another process
*Probe	- Determine if communication has been completed
*Barrier	- Synchronize processes
Put	- Put data from the master process onto remote processes
Get	- Put data from remote process onto the master process
Bcast	- Transmit data to all processes using a tree structure

Distribution

Distribute	- Distribute a matrix according to the values of Coord
Collect	- Collect a matrix according to the mask created by Distribute
Shift	- Shift data between processes

Arithmetic

Max	- Find the pointwise maximum of matrices on several processes
Min	- Find the pointwise minimum of matrices on several processes
Sum	- Find the pointwise sum of matrices on several processes
Prod	- Find the pointwise product of matrices on several processes

Graphics	
Window Reset	- Arrange figures in a grid according to function parameters - Reset default window position to MATLAB default
Window Refresh	- Repaint all current figures

Table 5. MultiMATLAB Commands

MultiMATLAB was evaluated and found to be useful for quick development of embarrassingly parallel (replication of the same algorithm on blocks of data) and SPMD parallel routines. The following sections discuss our findings resulting from the migration of test routines in Table 6 to the MultiMATLAB environment.

Test Code	Description	Problem Size	# Lines
3D	Generation of 3D-surface	XxYxZ = [51:51x31:31x21:21]	30
MAT_INV4	matrix inverse	X=Y=[100:800]	43
FFT	MATLAB column-wise FFT	X=Y=[100:800]	51
Simple	MATLAB FFT2 algorithm	X=Y=[100:2500]	40
MM	Matrix Multiply	X=Y=[50:800]	25

Table 6. MultiMATLAB Test Codes

3.3.1 Migrating MATLAB Code to MultiMATLAB and Matrix Multiply

To introduce our findings, the process of migrating source code to MultiMATLAB is discussed, beginning with our serial matrix multiply example in Figure 1.

```
% serialMul.m
% This MultiMATLAB M-file serial matrix multiplication
%
```

```
function z = serialMul (n);

A = randn(n);
B = randn(n);
tic;
  for i = 1 : n
    for j = 1 : n
      C( i, j ) = A( i, : ) * B( :, j ) ;
    end
  end
z = toc;
```

Figure 1. serialMul.m - serial matrix multiplication

Figure 2 illustrates how a master process broadcasts the matrices to the slave processors and calls a new routine, remMul.m in Figure 3, which performs the partitioned matrix multiply. As shown, this partitioning is performed on each processor according to limits set by values of "L" (Lower) and "U" (Upper) which are determined by the child process environment variable "Nproc." Upon completion, the submatrix computation on all slave processors in blocks "C" are returned to the master processor.

```
% matMul.m
% This MultiMATLAB M-file performs matrix multiplication of two ( n x n ) square
matrices % w/ random entries for scalability timing tests.
function z = matMul (n)
A = randn(n);
B = randn(n);
C = zeros(n);
tic;
Bcast( 'A' );
Bcast( 'B' );
Bcast( 'C' );

Eval( 'remMul' );
D = Sum( 'C' ) ;
```

Figure 2. MultiMATLAB version - Master Processor

```
% remMul.m
%
% Called by matMul.m to perform matrix multiplication on each processor for
% its own  slice of the resulting matrix.
%

L = ID * n / Nproc + 1 ;
U = (ID+1)* n / Nproc  ;

  for i = 1 : n
    for j = L : U
      C( i, j ) = A( i, : ) * B( :, j ) ;
    end
  end

%

% disp( 'done multiplying remotely' )
```

Figure 3. Slave Processor Execution

The matrix multiply example was easily implemented and found to yield measurable improvement for matrix sizes exceeding n=100, as indicated in Figure 4. The top trace represents the serial performance, the middle trace represents the performance with MultiMATLAB, and the lowest trace represents the communication overhead from MultiMATLAB. Other test codes, as discussed in the following sections, yielded varying results.

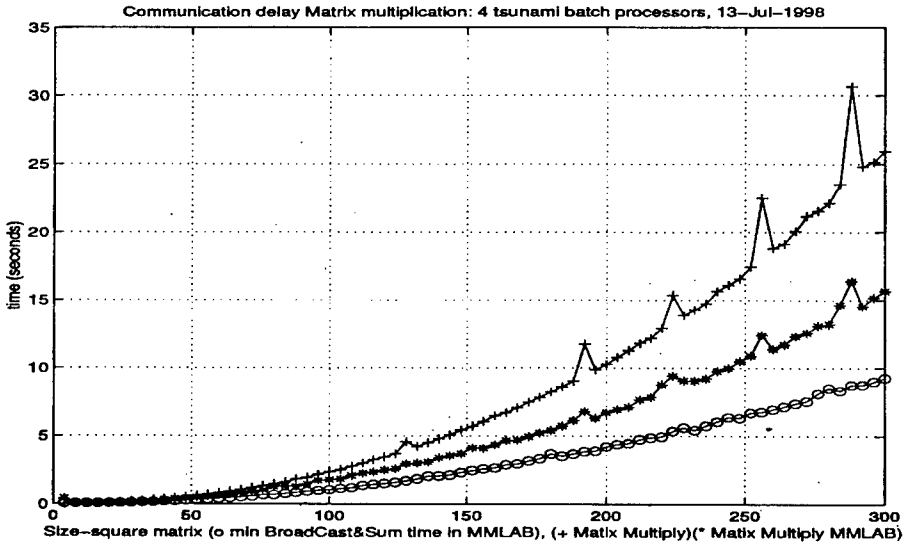


Figure 4. Matrix Multiply Performance with MultiMATLAB

3.3.2 3D

The "3D" test code generates eigenvectors of 3x3 matrices. Because the computation load exceeds the communication latency, MultiMATLAB improves the execution as the number of eigenvector calculations performed grows to 10^6.

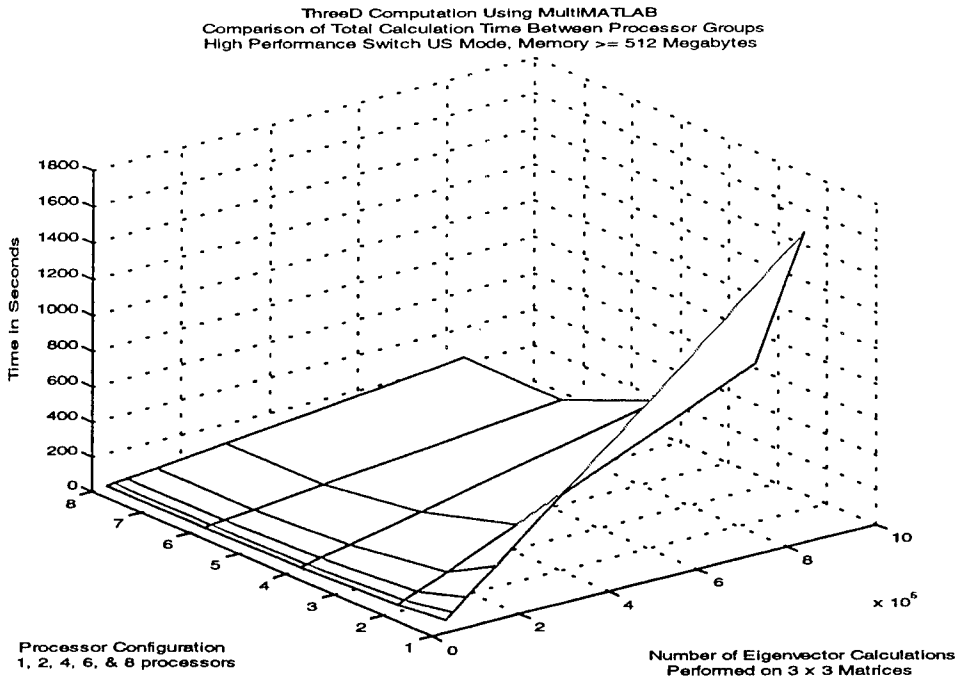


Figure 5. "3D" Performance with MultiMATLAB

3.3.3 Child Processor Collection Approach and Matrix Inversion

As MATLAB provides a library module for matrix inverse, we developed an implementation of matrix inverse computation [9] based on the formula:

$A^{-1} = 1/\det(A)*adj(A)$

where

det(A) is the matrix determinant of A and adj(A) is the adjoint of A.

However, because the matrix summation here utilized an ordered "for" loop, performance degraded with increased processors. As our first implementation utilized the

MultiMATLAB "Get" command, which orders data collection from child processors, our example experiences degraded performance, as indicated in Figure 6.

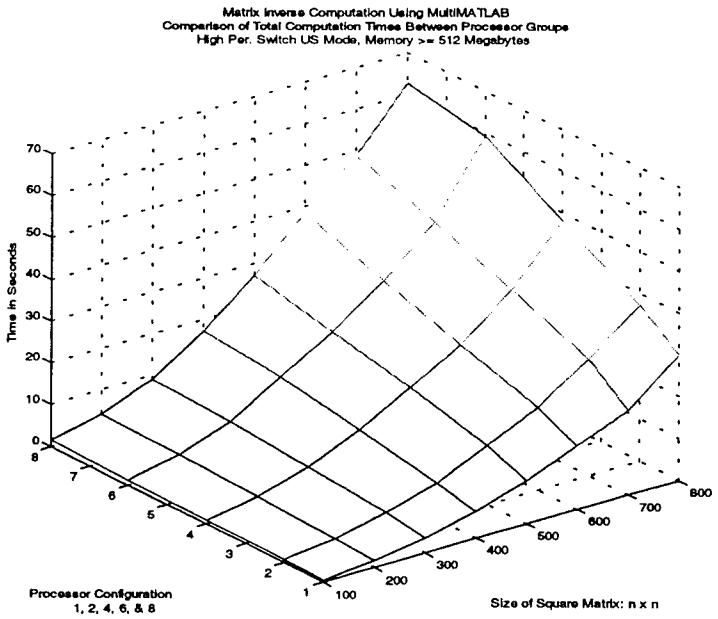


Figure 6. Matrix Inverse Performance with MultiMATLAB using "Get"

However, when our algorithm utilized the unordered "Sum" to collect matrix blocks from child processors, we achieved measured speedup with increased processors, as indicated in Figure 7.

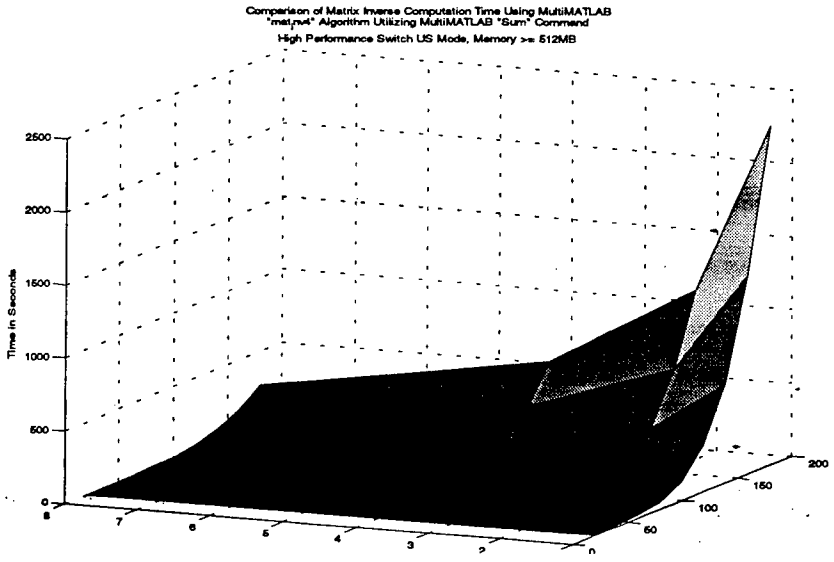


Figure 7. Matrix Inverse Performance with MultiMATLAB Using "Sum"

3.3.4 Fast Fourier Transform (FFT)

To validate our claim regarding the influence of child collection, we examined two FFT algorithms, a column-wise and two-dimensional matrix Fast Fourier Transforms. Though our column-wise algorithm, "Simple FFT," does have speedup at 2 processors, these experienced similar degradation because of the results gathering approach, as indicated in Figures 8 and 9.

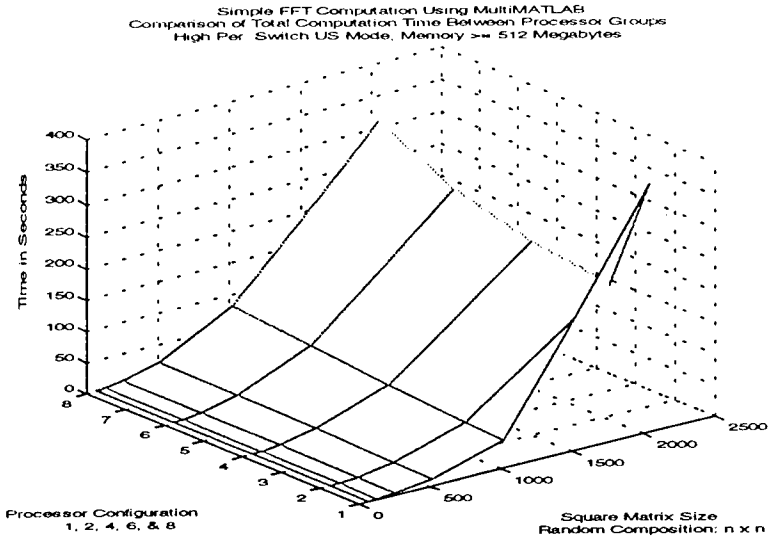


Figure 8. "Simple FFT" Performance with MultiMATLAB

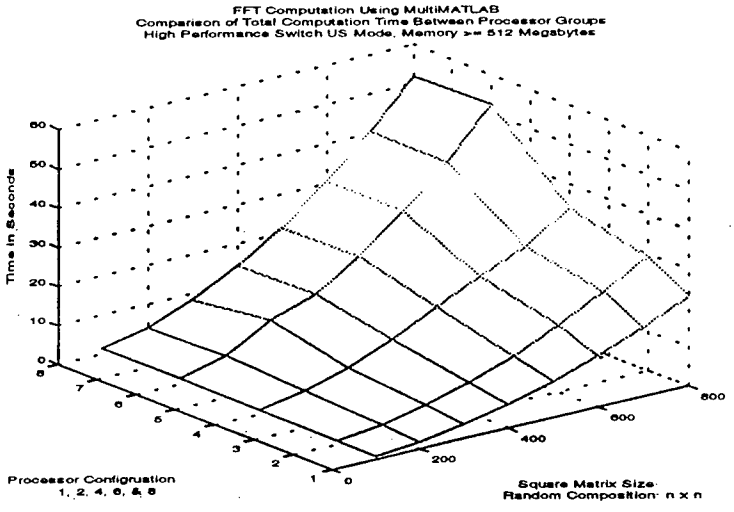


Figure 9. Column-wise 2D FFT Performance with MultiMATLAB

In summary, we found MultiMATLAB to be effective for scenarios in which the algorithm is a data parallel problem, and the developer has many MATLAB licences and little development time. However, though MultiMATLAB is simple to install and use, it is not yet commercially supported.

4 FULL SUITE APPROACH – Real Time (RT) Express

4.1 OVERVIEW

RTExpress™ performs automatic compilation to C and parallelization of software written in MATLAB. Parallelization of the application can be done automatically by RTExpress, explicitly directed by the developer, or a combination of both.

RTExpress' graphical user interface (GUI) Target Balancing Tool directs the interaction with the underlying software which control:

- Parallel communications
- Data acquisition and output (I/O)
- Real-time user displays and controls
- Real-time performance monitoring
- Post-mortem analysis

RTExpress utilizes and works in conjunction with the MATLAB compiler, mathematical subroutine libraries, parallel function libraries, BLAS and BLACS libraries, the native C compiler and MPI. Figure 10 depicts the function flow of the various RTExpress components.

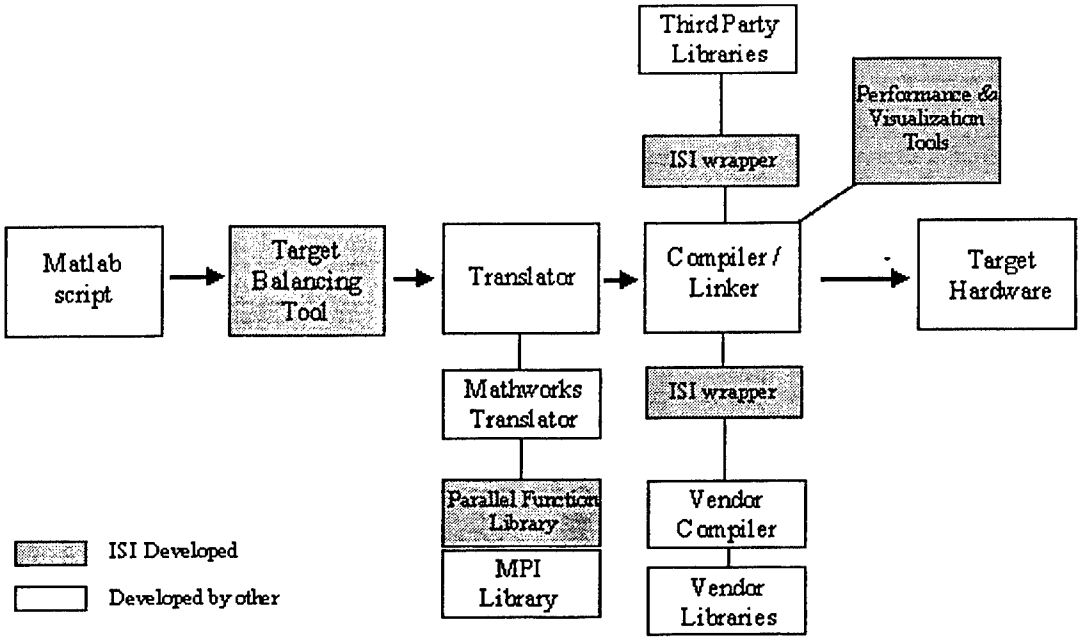


Figure 10. Function Flow of RTExpress Components

4.2 RTExpress – Usage and Dependencies

RTExpress can be used to translate a serial MATLAB application into a parallel application by several different methods which vary greatly in the amount of developer knowledge and effort required. They can also vary greatly in the efficiency of the resulting parallel application. The primary parallel paradigms RTExpress implements are described in this section.

4.2.1 Automatic Data Parallelism

In accordance with the data parallel model, the developer simply specifies the number of instances of the executable to be run and allows RTExpress to distribute the work across the node pool and produce the translated code with all of the necessary parallel constructs for data parallelism. RTExpress automatically distributes the matrix data and operations according to the number of instances without any developer interaction.

4.2.2 Function /Task Parallelism, Pipelined Parallelism

This involves decomposing a single serial application into multiple modules for concurrent execution. RTE recognizes lines of code which can be executed concurrently and partitions code into discreet groups. Each "group" is then viewed as a separate executable (task Inter-task communications for shared data is handled automatically by RTExpress by means of its `groupimport` and `groupexport` routines. RTExpress allows the simultaneous incorporation of both data and functional parallelism.

4.3 Evaluation of RTExpress

RTExpress was evaluated on site at the MHPCC in the areas of translation effectiveness to parallel models, speedup, scalability, portability and robustness. These along with other characteristics are detailed in this section.

Over a dozen codes were tested with RTExpress of which nine achieved significant results. Though the remaining codes did not operate under RTE, they did provide valuable insights into the product's behavior and limitations. A brief description of each of these codes appears in the Table 7.

Test Code	Description	#Lines*
`tst_icn.m`	Incomplete Cholesky factorization of matrix. Uses a double nested loop to compute the incomplete Cholesky factorization of a matrix. Main feature is use of a multi-typed built-in (sqrt). Problems size: 400x400	33
`tst_dirich.m`	Dirichlet solution to Laplace's equation. An iterative method for the solution to Laplace's equation. An elementary-operation intensive program which requires element-wise access of grid elements. Problem size: 41x41	39
`tst_finedif.m`	Finite difference solution to the wave equation. A numeric approximation method for the solution of hyperbolic differential equations. This is an elementary-operation intensive program that performs indexed updates to a two dimensional grid. Problem size: 451x451	28
`inv.m`	Inverse matrix operation performed on a 2000 x 2000 matrix. The MATLAB `inv()` function is used to perform the operation.	8
`loops.m`	Four simple loops, each of which is double nested, where i=2000 and j=2000. A simple assignment is made for each element in a 2000x2000 matrix.	51
`simple.m`	2D FFT operations. Using the MATLAB `fft()` and `ifft()` functions on a 1024x1024 matrix.	24
`filt.m`	Image filtering operations. An input file provides multiple frames of an initial image. Each frame is comprised of a 278x392 matrix of reals. An edge detection operation is performed on each frame. 10 frames are processed.	74

rawtofft1.m	Doppler Filter algorithm 1	32
rawtofft2.m	Doppler Filter algorithm 2	46

* # Lines = number of source code lines minus comment lines. White lines included.

Table 7. RTExpress Test Codes

All timings were performed in "batch" mode to ensure dedicated CPU usage. The IBM SP nodes used were P2SC, 160 MHz "thin" nodes with 512 MB of memory (2 x 256 MB memory cards). Data cache was 64 KB with 128 byte cache lines. Results shown are the average of a minimum of five independent runs per code. The MATLAB "tic" and "toc" calls were used to obtain the timings.

The legends for graphs should be interpreted from Table 8:

IP	RTExpress produced C language object module run with Internet Protocol communications.
US	RTExpress produced C language object module run with User Space communications.
MAT-I	Execution within MATLAB as interpreted source. Always run serially on one processor.
MAT-C	Execution within MATLAB as a MATLAB compiled C language object module. Always run serially on one processor.

Table 8. RTExpress Communication Variations

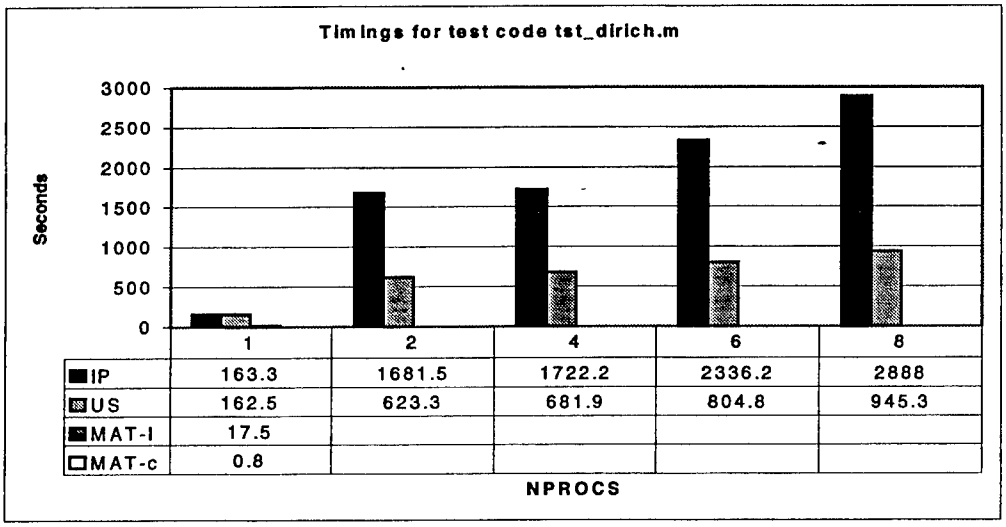
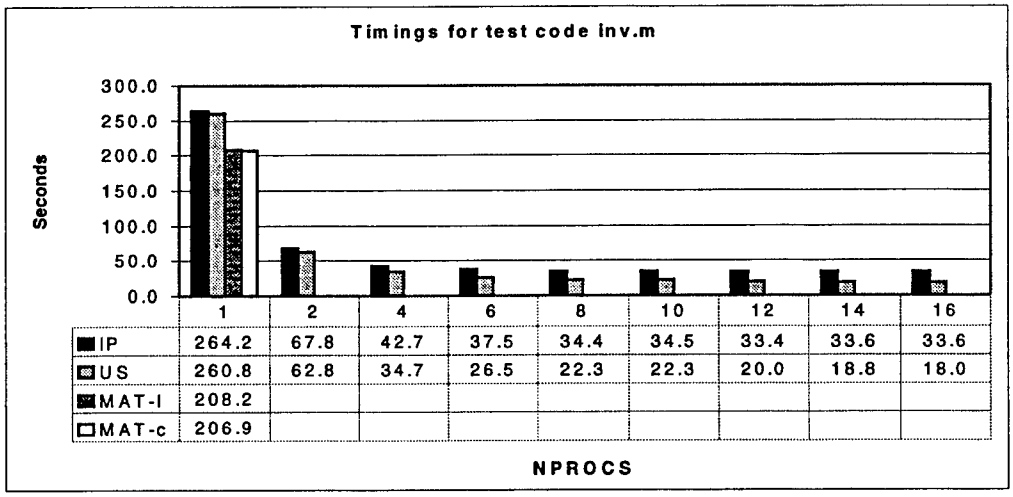


Figure 11. Timings for test code tst_dirich.m

This code exhibits a case where translation by RTExpress dramatically decreased the performance. Figure 11 shows that performance continued to decrease as more processors

were added. The sizeable differences between IP and US protocols is suggestive of a significant amount of data transfer taking place over the network between the multiple instances of the parallel program. Note that the MATLAB compiled executable demonstrated an execution time of less than one second when it was run serially on a single processor.

The `tst_dirich.m` code is an iterative method and contains multiple loops, one of which is double nested inside of a while loop. According to the vendor, RTExpress performs very poorly on loops, particularly nested loops, and suggested that codes should be written as much as possible using vector syntax instead.



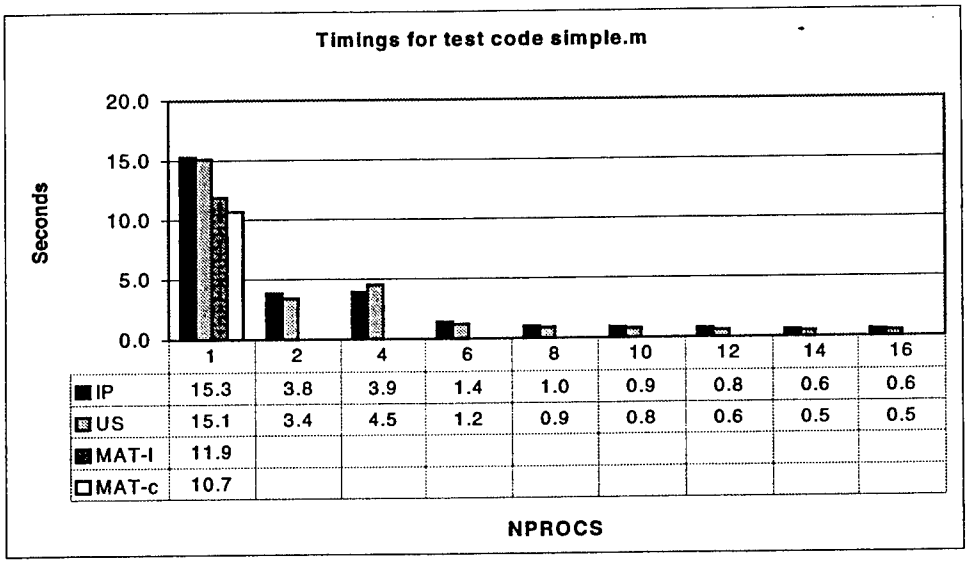
	1	2	4	6	8	10	12	14	16
IP	264.2	67.8	42.7	37.5	34.4	34.5	33.4	33.6	33.6
US	260.8	62.8	34.7	26.5	22.3	22.3	20.0	18.8	18.0
MAT-I	208.2								
MAT-c	206.9								

Figure 12. Timings for test code inv.m

The test code Inv.m consists of a call to the MATLAB `inv()` function and demonstrates considerable improvement with RTExpress, as indicated in Figure12. The matrix size was 2000 x 2000 real elements, approximately 32 MB in size, making it a reasonably large data set. Note that on one processor, the MATLAB compiler only marginally improved the interpreted execution. Also note the overhead imposed by RTExpress for the execution one processor.

Especially worth noting is the 400+ % speedup produced when the RTExpress executable is run on 2 processors versus 1. Adding additional processors continues to improve performance, though the results suggest that any gains achieved beyond 6 processors does not significantly improve performance. The relatively small differences between IP and US timings, considering that US is generally three times faster than IP, indicates that there is little interprocess communications.

The effort to parallelize this code was minimal, relying upon RTExpress to automatically analyze and implement the data decomposition.

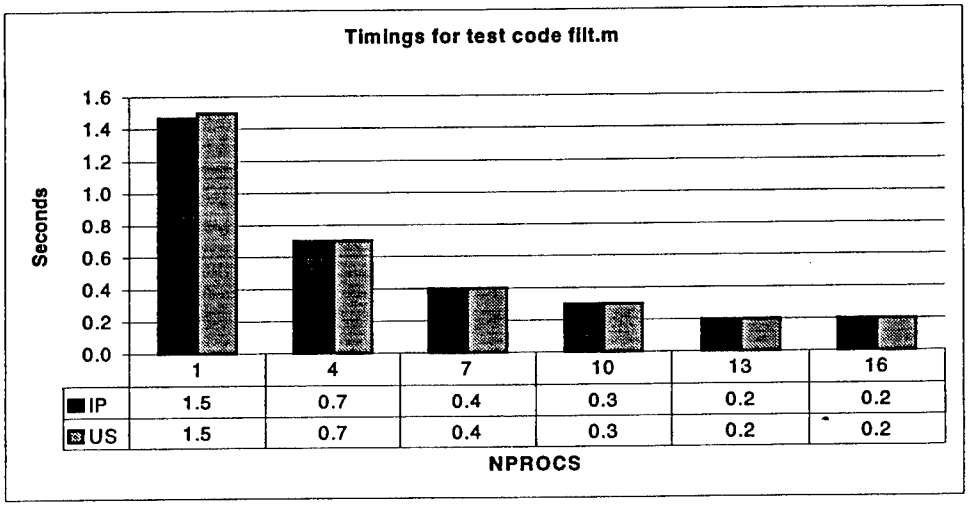


	1	2	4	6	8	10	12	14	16
IP	15.3	3.8	3.9	1.4	1.0	0.9	0.8	0.6	0.6
US	15.1	3.4	4.5	1.2	0.9	0.8	0.6	0.5	0.5
MAT-I	11.9								
MAT-c	10.7								

Figure 13. Timings for test code simple.m

The "simple.m" test code, as shown in Figure 13, illustrates RTExpress performance for an "embarrassingly parallel" code. RTExpress implements the FFT operations using data distribution of the matrix by columns to the number of available processes. As shown, very good speedup is achieved with 2 processors and weakened improvement after 6 processors.

Again, the effort to parallelize this code was minimal, relying upon RTExpress to automatically analyze and implement the data decompostion.

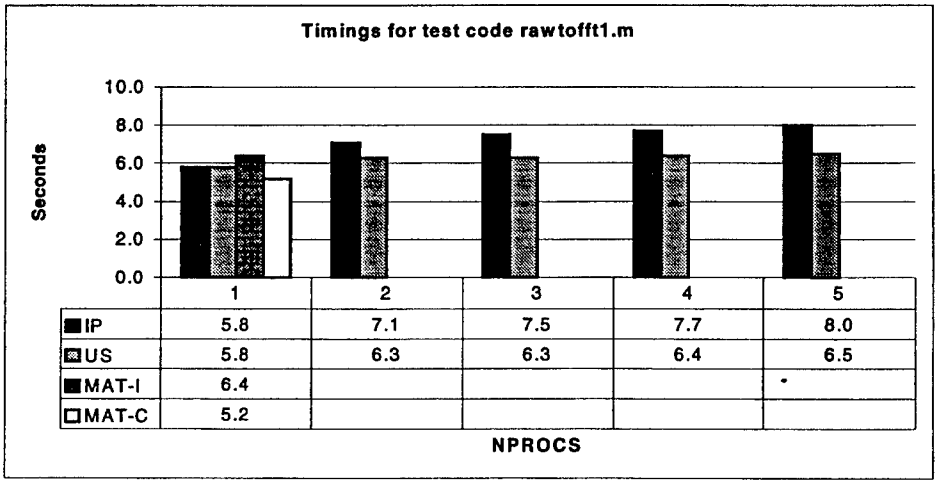


	1	4	7	10	13	16
IP	1.5	0.7	0.4	0.3	0.2	0.2
US	1.5	0.7	0.4	0.3	0.2	0.2

Figure 14. Timings for test code filt.m

The `filt.m` evaluation code utilized the mixed-mode parallel paradigm of RTExpress. The original serial m-file was partitioned into four distinct groups using the RTExpress Target Balancing Tool editor. The first group continually reads a 5.5 MB input file consisting of multiple frames of image data stored as individual matrices. The other groups work in parallel to perform an image processing operation on the matrix, in this case, edge detection. The edge detection algorithm code was easily decomposed into three functional tasks. As each frame is read by the first group, the data is passed to each of the other compute groups, which then operate independently. This functional/task parallelism is enhanced by adding additional processors to the compute tasks, allowing them to perform data parallelism within their group.

Figure 14 demonstrates improvement obtained by adding additional processors to each of the edge detection groups. For example, when NPROCS is 4, then the "read" group has one processor, as does each of the three compute groups. When NPROCS is 7, the "read" task still has one processor, but each of the compute groups now has two processors. Because I/O is an inherently serial operation in RTExpress, there is no benefit to using more than one processor with the "read" group. During this evaluation, compute groups were always kept equal in the number of processors assigned to them. Additional testing could certainly be done to determine execution behavior when the processors within a group vary.



	1	2	3	4	5
IP	5.8	7.1	7.5	7.7	8.0
US	5.8	6.3	6.3	6.4	6.5
MAT-I	6.4				
MAT-C	5.2				

Figure 15. Timings for test code rawtofft1.m

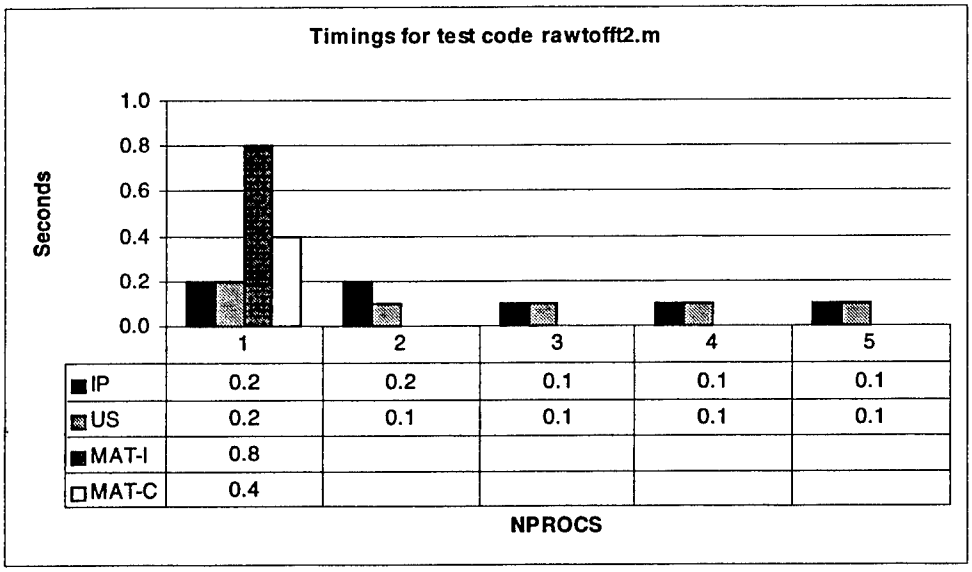


Figure 16. Timings for test code rawtofft2.m

The two evaluation codes in Figures 15 and 16 performed the Fast Fourier Transforms utilizing opposing algorithms. The first code demonstrated weaker parallelization due to a heavy utilization of looping control.

The "loops.m" test code further demonstrate the RTE's dependency on MATLAB loop control. In the serial version of the code, 4 loops are executed sequentially. Each code fragment consists of a doubly nested loop over a 2000x2000 matrix which performs a simple assignment to each element in the matrix.

Timings for Test Code loops.m (non-vector)	
MATLAB interpreted. All 4 loops executed serially on a single processor.	477.8
MATLAB compiled. All 4 loops executed serially on a single processor.	16.2
RTExpress. All 4 loops executed serially on a single processor	596.5
RTExpress. Parallel. One loop distributed to each of 4 processors. IP communications	601.3
RTExpress. Parallel. One loop distributed to each of 4 processors. US communications	601.7

Table 9. Timings for test code loops.m (non-vector)

As shown in Table 9, RTExpress parallel execution took 4 times a long as the RTExpress serial execution (4 processors x 601 seconds). Upon discovery the developers of RTExpress advised rewriting the algorithm utilizing vector operation. Following this advice, as shown in Figure 17, yielded considerable speedup as shown in Table 10.

Before: non-vector	After: vectorized

A = zeros(2000,2000);
for i = 1:2000
  for j = 1:2000
  m = i;
  n = j;
  A(i,j) = m*n;
  End
End
	A = zeros(2000,2000);
	l = linspace(1,2000,2000);
	for i = 1:2000
	  m = i;
	  A(i,:) = m .* l;
	end

Figure 17. Vectorized and Non-Vectorized Versions of Matrix Multiply

Timings for Test Code loops.m (vectorized)	
MATLAB interpreted. All 4 loops executed serially on a single processor.	12.4
MATLAB compiled. All 4 loops executed serially on a single processor.	10.8
RTExpress. All 4 loops executed serially on a single processor.	37.2
RTExpress. Parallel. One loop distributed to each of 4 processors. IP communications	9.4
RTExpress. Parallel. One loop distributed to each of 4 processors. US communications	9.3

Table 10. Timings for test code loops.m (vectorized)

The "loops.m" example here provides insight into the overhead imposed by RTEpress by comparison with the times for the compiled (10.8s) and RTExpress operation (37.2s) on one processor.

The test codes listed in Table 11 further illustrate performance degradation with resulting from nested loops.

Timings for Test Code tst_icn.m and tst_finedif.m		
	tst_icn.m	tst_finedif.m
MATLAB interpreted	11.4	11.8
MATLAB compiled	0.2	0.5
RTExpress serial w/IP communications	99.6	153.6
RTExpress serial w/US communications	95.9	153.3
RTExpress parallel - 2, 4, 8, 16 processors	All greater than 7200	All greater than 7200

Table 11. Timings for test code tst_icn.m and tst_finedif.m

5 RECOMMENDATIONS

As mentioned, the degree of improvement achieved by each of these utilties is dependent on many factors such as MATLAB code structure, communication interface, and execution environment. In addition, there are other factors users need to consider such as number of MATLAB licenses available, amount of development time available, along with availability and support, which factor into using one of the mentioned parallel-MATLAB utilities. As such, we provide recommendations for both users and developers for each of the classes of approaches.

Compiler Approaches, utility developers

- **MathTools, MathWorks:** The Falcon report demonstrated two primary areas which would make both MATCOM and MathWorks compilers better products: dynamic inference and inference phase comparisons. Utilizing both of these techniques will improve these commercial products.
- **MathWorks:** MATLAB's latest compiler version V2.0.1 does not allow any manual inference customization, this being solely handled by the MATLAB compiler. It was shown here and in the Falcon report that inference control can yield measurable performance differences. In short, allow users the control of inference by returning the use of the –r and –i switches and pragmas.

Compiler Approaches, users:

The following is a list of recommendations that developers should keep in mind when using the compilers approaches:

- MATLAB programs which spend a significant time performing library calls will not be greatly improved. Use of the MATLAB "tic" and "toc" commands can quickly determine the location of your heavy computations.
- MATLAB routines with significant matrix accesses will benefit greatly from intelligent shape inference and matrix preallocation. Therefore, if you don't require variable matrix size, preallocate your matrices.
- MATLAB "for" loops should be vectorized as much as possible.

- Small MATLAB routines should be written inline (same file or subroutine) as much as possible.
- Identify your target compiled language data types at development time. Simply put comments around your data structures indicating your estimate of what the data types would be in a compiled language.

Interpretive Approaches, utility developers:

- As the Parallel Toolbox (PT) product requires a MATLAB license for each processor, and it is not distributed by a commercial entity it is not recommended as a general purpose solution.
- Matpar and MultiMATLAB complement each other in that Matpar requires only one license and utilizes high performance libraries. Further, both incorporate the defacto High Performance Computing communication mechanism, message passing. It is our recommendation to the developers that that the technologies be encapsulated and distributed together.

Interpretive Approaches, users:

- Use of the interpretive approaches is with risk in that none of the approaches discussed are commercially maintained and distributed.
- Users need to closely manage the computation/communication ratio in that all of the solutions investigated are built on slow communication interfaces.
- The MultiMATLAB and Matpar utilities are valuable in that they require less time to implement a solution and the user stays within the familiar MATLAB environment for all stages of development.
- Do not use the MultiMATLAB "Get" call to collect data from worker nodes. The time spent in sequencial waits will overwhelm any savings from a Massively Parallel distribution.

Full Suite approach, utility developers:

- RTExpress utilizes MATLAB's mcc compiler which has limitations listed above. RTE may benefit from by allowing the use of either MathWork's or MathTool's compilers.
- RTExpress hides the control of mcc v1.2 compiler switch settings (-r-I). When mcc returns manual editing of variable inference, RTExpress should pass this capability up to the developer.

Full Suite approach, users:

- RTExpress does require some training, but the potential for MATLAB improvement can be great depending on your code size and time available.
- Vectorize loops whenever possible.

APPENDIX A - SOURCE MATLAB TEST CODE

This section lists all of the source MATLAB codes used for the on-site testing.

A1AQ

```
function [SRmat,quad,err] =
tst_adapt(a,b,sz_guess,tol)
%
% Sample call
%   [SRmat,quad,err] = adapt('f',a,b,tol)
% Inputs
%   f       name of the function
%   a       left  endpoint of [a,b]
%   b       right endpoint of [a,b]
%   tol   convergence tolerance
% Return
%   SRmat   matrix of adaptive Simpson quadrature
values
%   quad    adaptive Simpson quadrature
%   err     error estimate
%
% NUMERICAL METHODS: MATLAB Programs, (c) John H.
Mathews 1995
% To accompany the text:
% NUMERICAL METHODS for Mathematics, Science and
Engineering, 2nd Ed, 1992
% Prentice Hall, Englewood Cliffs, New Jersey, 07632,
U.S.A.
% Prentice Hall, Inc.; USA, Canada, Mexico ISBN 0-13-
624990-6
% Prentice Hall, International Editions:   ISBN 0-13-
625047-5
% This free software is compliments of the author.
% E-mail address:      in%"mathews@fullerton.edu"
%
% Algorithm 7.5 (Adaptive Quadrature Using Simpson's
Rule).
% Section       7.4, Adaptive Quadrature, Page 389
%------------------------------------------------------

SRmat = zeros(sz_guess,6);
iterating = 0;
done = 1;
% SRvec = zeros(6); not necessary.

%% SRvec = my_srule('f',a,b,tol);

h  = (b - a)/2;
c = (a + b)/2;
Fa = 13.*(a - a.^2).*exp(-3.*a./2); %f(a);
Fc = 13.*(c - c.^2).*exp(-3.*c./2); %f(c);
Fb = 13.*(b - b.^2).*exp(-3.*b./2); %f(b);
S  = h*(Fa + 4*Fc + Fb)/3;
S2 = S;
tol1 = tol;
err  = tol;
SRvec = [a b S S2 err tol1];

SRmat(1,1:6) = SRvec;
m = 1;
state = iterating;
while (state == iterating)
  n = m;
  for j=n:-1:1,
    p = j;
    SR0vec = SRmat(p,:);
    err = SR0vec(5);
    tol = SR0vec(6);
    if  (tol <= err),
      state = done;
      SR1vec = SR0vec;
      SR2vec = SR0vec;
      a = SR0vec(1);
      b = SR0vec(2);
      c = (a + b)/2;
      err = SR0vec(5);
      tol = SR0vec(6);
      tol2 = tol/2;
%      SR1vec = my_srule('f',a,c,tol2);
      a0 = a;
      b0 = c;
      tol0 = tol2;
      h  = (b0 - a0)/2;
      c0 = (a0 + b0)/2;
      Fa = 13.*(a0 - a0.^2).*exp(-3.*a0./2); %f(a0);
      Fc = 13.*(c0 - c0.^2).*exp(-3.*c0./2); %f(c0);
      Fb = 13.*(b0 - b0.^2).*exp(-3.*b0./2); %f(b0);
      S  = h*(Fa + 4*Fc + Fb)/3;
      S2 = S;
      tol1 = tol0;
      err1  = tol0;
      SR1vec = [a0 b0 S S2 err1 tol1];

% SR2vec = my_srule('f',c,b,tol2);

      a0 = c;
      b0 = b;
      tol0 = tol2;
      h  = (b0 - a0)/2;
      c0 = (a0 + b0)/2;
      Fa = 13.*(a0 - a0.^2).*exp(-3.*a0./2); %f(a0);
      Fc = 13.*(c0 - c0.^2).*exp(-3.*c0./2); %f(c0);
      Fb = 13.*(b0 - b0.^2).*exp(-3.*b0./2); %f(b0);
      S  = h*(Fa + 4*Fc + Fb)/3;
      S2 = S;
      tol1 = tol0;
      err1  = tol0;
      SR2vec = [a0 b0 S S2 err1 tol1];

      err = abs(SR0vec(3)-SR1vec(3)-SR2vec(3))/10;
      if  (err < tol),
        SRmat(p,:) = SR0vec;
        SRmat(p,4) = SR1vec(3) + SR2vec(3);
        SRmat(p,5) = err;
      else
        SRmat(p+1:m+1,:) = SRmat(p:m,:);
        m = m+1;
        SRmat(p,:) = SR1vec;
        SRmat(p+1,:) = SR2vec;
        state = iterating;
      end
    end
  end
end
quad = sum(SRmat(:,4));
err = sum(abs(SRmat(:,5)));
SRmat = SRmat(1:m,1:6);
```

A.2 CQ

```
%
% Copyrighted, 1993, by Richard Barrett, Michael
Berry,
% Tony Chan, James Demmel, June Donato, Jack Dongarra,
Victor Eijkhout,
% Roldan Pozo, Charles Romine, and Henk van der Vorst.
%
% The M-file was created to supplement "Templates for
the Solution of
% Linear Systems:  Building Blocks for Iterative
Methods," by Richard Barrett,
% Michael Berry, Tony Chan, James Demmel, June Donato,
Jack Dongarra, Victor
% Eijkhout, Roldan Pozo, Charles Romine, and Henk van
der Vorst (SIAM, 1994).
%
% You are free to modify any of the files and create
new functions, provided
% that you acknowledge the source in any publication
and do not sell the
% modified file.
%

function [x, flag, Error, iter] = tst_cgopt(A, b,
max_it, tol, x)

%  -- Iterative template routine --
%     Univ. of Tennessee and Oak Ridge National
Laboratory
%     October 1, 1993
%     Details of this algorithm are described in
"Templates for the
%     Solution of Linear Systems: Building Blocks for
Iterative
%     Methods", Barrett, Berry, Chan, Demmel, Donato,
Dongarra,
%     Eijkhout, Pozo, Romine, and van der Vorst, SIAM
Publications,
%     1993. (ftp netlib2.cs.utk.edu; cd linalg; get
templates.ps).
%
%  [x, error, iter, flag] = cg(A, x, b, M, max_it,
tol)
%
% cg.m solves the symmetric positive definite linear
system Ax=b
% using the Conjugate Gradient method with
preconditioning.
%
% input   A      REAL symmetric positive definite
matrix
%         x      REAL initial guess vector
%         b      REAL right hand side vector
%         M      REAL preconditioner matrix (LDR -
removed from original M-file)
%         max_it INTEGER maximum number of iterations
%         tol    REAL error tolerance
%
% output  x      REAL solution vector
%         error  REAL error norm
%         iter   INTEGER number of iterations
performed
%         flag   INTEGER: 0 = solution found to
tolerance
%                        1 = no convergence given
max_it
%
% Modified by Luiz A. De Rose (derose@cs.uiuc.edu)
%

M = diag(diag(A));

flag = 0;                                    %
initialization
iter = 0;
bnrm2 = norm( b );
if  ( bnrm2 == 0.0 ), bnrm2 = 1.0; end
r = b - A*x;
Error = norm( r ) / bnrm2;
if ( Error < tol ), return, end
for iter = 1:max_it                          % begin
iteration
%  z = M \ r;
   z = r ./ diag(M);
   rho = (r'*z);
   if ( iter > 1 ),                          % direction
vector
      beta = rho / rho_1;
      p = z + beta*p;
   else
      p = z;
   end

   q = A*p;
   alpha = rho / (p'*q );
   x = x + alpha * p;                        % update
approximation vector

   r = r - alpha*q;                          % compute
residual
   Error = norm( r ) / bnrm2;                % check
convergence
   if ( Error <= tol ), break, end

   rho_1 = rho;

end

if ( Error > tol ), flag = 1; end            % no
convergence
```

A3FD

```
function U = tst_finedif(a,b,c,n,m)
%-------------------------------------------------
%FINEDIF   Finite difference solution to the wave
equation.
% Sample call
%   U = finedif('f','g',a,b,c,n,m)
% Inputs
%   f   name of a boundary function
%   g   name of a boundary function
%   a   is the width of interval [0 a]: 0<=x<=a
%   b   is the width of interval [0 b]: 0<=t<=b
%   c   is the constant in the wave equation
%   n   is the number of grid points over [0 a]
%   m   is the number of grid points over [0 b]
% Return
%   U   solution: matrix
%
% NUMERICAL METHODS: MATLAB Programs, (c) John H.
Mathews 1995
% To accompany the text:
% NUMERICAL METHODS for Mathematics, Science and
Engineering, 2nd Ed, 1992
% Prentice Hall, Englewood Cliffs, New Jersey, 07632,
U.S.A.
% Prentice Hall, Inc.; USA, Canada, Mexico ISBN 0-13-
624990-6
```

```
% Prentice Hall, International Editions:   ISBN 0-13-
625047-5
% This free software is compliments of the author.
% E-mail address:       in%"mathews@fullerton.edu"
%
% Algorithm 10.1 (Finite-Difference Solution for the
Wave Equation).
% Section 10.1, Hyperbolic Equations, Page 507
%-------------------------------------------------------
nm1 = n - 1;
h = a/nm1;
k = b/(m-1);
r = c*k/h;
r2 = r^2;
r22 = r^2/2;
s1 = 1 - r^2;
s2 = 2 - 2*r^2;
U = zeros(n,m);
for i=2:nm1,
  x = h*(i-1);
  h_im2 = h*(i-2);
  U(i,1) = sin(pi*x) + sin(3*pi*x);
  U(i,2) = s1*(sin(pi*x) + sin(3*pi*x)) + ...
           r22*(sin(pi*h*(i)) + sin(3*pi*h*(i))) + ...
            sin(pi*h*(i-2)) + sin(3*pi*h*(i-2));
end
for j=3:m,
  for i=2:nm1,
    U(i,j) = s2*U(i,j-1) + r2*(U(i-1,j-1) + U(i+1,j-
1)) - U(i,j-2);
  end
end
```

A4DI

```
function U = tst_dirich(f1,f2,f3,f4,a,b,h,tol,max1)
%-------------------------------------------------------
%DIRICH  Dirichlet solution to Laplace's equation.
% Sample call
%   U = dirich('f1','f2','f3','f4',a,b,h,tol,max1)
% Inputs
%   f1      name of a boundary function
%   f2      name of a boundary function
%   f3      name of a boundary function
%   f4      name of a boundary function
%   a       width of interval [0 a]: 0<=x<=a
%   b       width of interval [0 b]: 0<=y<=b
%   h       step size
%   tol     convergence tolerance
%   max1    maximum number of iterations
% Return
%   U       solution: matrix
%
% NUMERICAL METHODS: MATLAB Programs, (c) John H.
Mathews 1995
% To accompany the text:
% NUMERICAL METHODS for Mathematics, Science and
Engineering, 2nd Ed, 1992
% Prentice Hall, Englewood Cliffs, New Jersey, 07632,
U.S.A.
% Prentice Hall, Inc.; USA, Canada, Mexico ISBN 0-13-
624990-6
% Prentice Hall, International Editions:   ISBN 0-13-
625047-5
% This free software is compliments of the author.
% E-mail address:       in%"mathews@fullerton.edu"
%
% Algorithm 10.4 (Dirichlet Method for Laplace's
Equation).
% Section 10.3, Elliptic Equations, Page 531
%-------------------------------------------------------
n = fix(a/h)+1;
m = fix(b/h)+1;
ave = (a*(f1+f2) +  b*(f3+f4))/(2*a+2*b);
U = ave*ones(n,m);
for j=1:m
  U(1,j) = f3;
  U(n,j) = f4;
end
for i=1:n
  U(i,1) = f1;
  U(i,m) = f2;
end
U(1,1) = (U(1,2) + U(2,1))/2;
U(1,m) = (U(1,m-1) + U(2,m))/2;
U(n,1) = (U(n-1,1) + U(n,2))/2;
U(n,m) = (U(n-1,m) + U(n,m-1))/2;
w = 4/(2+sqrt(4-(cos(pi/(n-1))+cos(pi/(m-1)))^2));
err = 1;
cnt = 0;
while ((err>tol)&(cnt<=max1))
  err = 0;
  for j=2:(m-1),
    for i=2:(n-1),
      relx = w*(U(i,j+1)+U(i,j-1)+U(i+1,j)+ U(i-1,j)-
4*U(i,j))/4;
      U(i,j) = U(i,j) + relx;
      if (err<=abs(relx))
         err=abs(relx);
      end
    end
  end
  cnt = cnt+1;
end
```

A.5 Ga

```
function [theta, phi, free] = tst_galrkn(N, rho,
Nplot)
%
% copyrighted, 1993, by Alejandro Garcia
%
% The routine supplement the book,
% "Numerical Methods for Physics Using MATLAB"
% (Prentice Hall).
%
% 2-dimensions using Galerkin method (Neumann boundary
cond.)
eps0 = 8.8542e-12;   % Permittivity (C^2/(N m^2))
L = 1;      % System size
M=2;        % Number of charges (M=2 is dipole)
% Initialize position and charge of line charges
d = 0.1*L;  % Dipole separation
xq(1) = L/2;
yq(1) = L/2+d/2;
q(1) = 1;
xq(2) = L/2;
yq(2) = L/2-d/2;
q(2) = -q(1);
a = zeros(N);
delt = ones(N,1);     % delt(i) = 1 + delta(i,1)
delt(1) = 2;
for k=1:M     % Sum over charges
 tempx = cos((0:N-1)*pi*xq(k)/L);
 tempy = cos((0:N-1)*pi*yq(k)/L);
 for i=1:N
  for j=1:N
    a(i,j) = a(i,j) + q(k)*tempx(i)*tempy(j)...
             /( ((i-1)^2+(j-1)^2 +
eps)*delt(i)*delt(j) );
  end
 end
```

```
end
a = 4/(eps0*pi^2) * a;   % Throw in the factor out in
front
phi = zeros(Nplot,1);
theta = pi * (0:Nplot-1)/(Nplot-1);
for k=1:Nplot
  x = L/2 + rho*sin(theta(k));     % Coordinates at
which to
  y = L/2 + rho*cos(theta(k));     % evaluate potential
  for i=1:N
   % tempx=cos((i-1)*pi*x/L);
   xtemp = cos((i-1)*pi*x/L);
   for j=1:N
     % phi(k) = phi(k) + a(i,j)*tempx*cos((j-
1)*pi*y/L);
     phi(k) = phi(k) + a(i,j)*xtemp*cos((j-1)*pi*y/L);
   end
  end
  % Plot potential and compare with free dipole
  r_rc = [rho*sin(theta(k))  rho*cos(theta(k))];
  Tmn = r_rc - [0  d/2];
  Tpl = r_rc + [0  d/2];
  free(k) = -q(1)/(2*pi*eps0)*(log(norm(Tmn)) -
log(norm(Tpl)));
% free(k) = -q(1)/(2*pi*eps0)*(log(norm(r_rc - [0
d/2])) - ...
%               log(norm(r_rc + [0  d/2])));
end
```

A6Ec

```
function [thplot, rplot, kinetic, potential, tplot,
totalE] = ...
        tst_orbec(r0, v0, tau, nstep);
%
% copyrighted, 1993, by Alejandro Garcia
%
% The routine supplement the book,
% "Numerical Methods for Physics Using MATLAB"
% (Prentice Hall).
%
% orbe - Program to compute the orbit of a comet
% using the Euler method.
% clear;  help orbe;  % Clear memory and print header
r = [r0 0];
v = [0 v0];
GM = 4*pi^2;       % Grav. const. * Mass of Sun
(au^3/yr^2)
mass = 1.;         % Mass of projectile
%%%%% MAIN LOOP %%%%%%
time = 0;
for istep=1:nstep
  rplot(istep) = norm(r);        % Record orbit for
polar plot
  thplot(istep) = atan2(r(2),r(1));
  tplot(istep) = time;
  kinetic(istep) = .5*mass*norm(v)^2;  % Record
energies
  potential(istep) = - GM*mass/norm(r);
  % Calculate new position and velocity
  accel = -GM*r/norm(r)^3;   % Gravity
  v = v + tau*accel;
  r = r + tau*v;        % Euler-Cromer step
  time = time + tau;
end
totalE = kinetic + potential;
```

A7RK

```
function xout = rk4_orb(x,t,tau,param)
%
% copyrighted, 1993, by Alejandro Garcia
%
% The routine supplement the book,
% "Numerical Methods for Physics Using MATLAB"
% (Prentice Hall).
%
%  Runge-Kutta integrator (4th order)
% Input arguments -
%   x = current value of dependent variable
%   t = independent variable (usually time)
%   tau = step size (usually timestep)
%   derivsRK = right hand side of the ODE; derivsRK is
the
%            name of the function which returns dx/dt
%            Calling format derivsRK(x,t,param).
%   param = extra parameters passed to derivsRK
% Output arguments -
%   xout = new value of x after a step of size tau
half_tau = 0.5*tau;
%F1 = feval(derivsRK,x,t,param);
F1 = gravrk(x,t,param);
t_half = t + half_tau;
xtemp = x + half_tau*F1;
%F2 = feval(derivsRK,xtemp,t_half,param);
F2 = gravrk(xtemp,t_half,param);
xtemp = x + half_tau*F2;
%F3 = feval(derivsRK,xtemp,t_half,param);
F3 = gravrk(xtemp,t_half,param);
t_full = t + tau;
xtemp = x + tau*F3;
%F4 = feval(derivsRK,xtemp,t_full,param);
F4 = gravrk(xtemp,t_full,param);
xout = x + tau/6.*(F1 + F4 + 2.*(F2+F3));
return;
```

A8IC

```
%
%  Incomplete Cholesky factorization of matrix A,
%  with the same sparsity pattern as A.  ICCG(0).
%
%     1 December 1993
%     R. Bramley
%     Department of Computer Science
%     Indiana University
%
%
function [L,Error] = tst_icn(A);

n = size(A,1);
L = A;
Error = 0;
for j = 1:n
  s = 0;
  for k = 1:j-1;
    s = s + L(j,k)*L(j,k);
  end
  r = sqrt(L(j,j) - s);

  if (r <= 0),
    Error = j;
    L(j,j) = 1;
  else
    L(j,j) = r;
  end  %  if (L(j,j) <= 0),

  t = 1/L(j,j);

  for i = j+1:n;
```

```
    if (L(i,j) ~= 0),
      s = 0;
      for k = 1:j-1;
        s = s + L(i,k)*L(j,k);
      end;
      L(i,j) = t*(L(i,j) - s);
    end;  % if (L(i,j) ~= 0)
  end;

end;  % for j
L = tril(L);
```

A.9 3D

```
%
%  Generation of a three dimensional surface
%
%     Mei-Qin Chen
%     The Citadel
%

function [c,b,dd,ind] = tst_3D(amin, amax, bmin, bmax,
cmin, cmax, h)
a=amin:h:amax;
b=bmin:h:bmax;
c=cmin:h:cmax;
na=length(a);
nb=length(b);
nc=length(c);
dd=zeros(nb,nc);
ind=0;
for kk=1:na,
    for ii=1:nb;
        for jj=1:nc;
            amat=[a(1,kk) b(1,ii) c(1,jj);1 0 0;0 1
0];
            ev=eig(amat);
            znorm=real(ev).^2+imag(ev).^2;
            if max(znorm)<1,
               ind=ind+1;
               dd(ii,jj)=a(1,kk);
            end;
        end;
    end;
end;
% surf(c,b,dd)
```

A.10 Simple

```
% generate data
%
a       = 1000 .* rand(1024,1024);
tic;
%
% compute 2-D FFT in separate steps
%
b       = fft(a);
c       = fft(b.');
d       = c.';
f       = ifft(d);
g       = ifft(f.');
h       = g.';
%
% compute performance
%
ttime   = toc
fflops  = 11665664
mflops  = fflops ./ ttime
%
% check result
%
err     = max(max(abs(a - h)))
```

A.11 FILT

```
function filt1

raw     = figure(2);
colormap(gray);
edg     = figure(3);
colormap(gray);

gdata = fopen('dummygray.bin','r');

hk = [-1 -1 -1; 0 0 0; 1 1 1];
vk = [-1 0 1; -1 0 1; -1 0 1];

threshe = 100.0;

while (1)

        [imgin,count] =
fread(gdata,[278,392],'uchar');
        if (count == 0)
                fclose(gdata);
                break;
        else

                figure(raw);
                image(imgin);

                img = imgin - mean(mean(imgin));


                hf = abs(conv2(img, hk, 'same'));

                vf = abs(conv2(img, vk, 'same'));

                cf = max(threshe, (hf + vf)) -
threshe;
                figure(edg);
                image(cf);
        end
end;
```

A.12 RawToFFT1

```
function [dcr] = rawToFFT1(x, numP, numR, numC)

%
%       range weight data
%

range_offset      = 116;
inv_denom = 1 ./ (32768*range_offset*range_offset);
for range = 1:numR,
   x((range-1)*numP+1:range*numP,:)=...
     x((range-
1)*numP+1:range*numP,:)*(range_offset+range-1)...
       *(range_offset+range-1)*inv_denom;
end;

%
%       compute doppler weighting matrix
%

w       = zeros(numP-1, numC);
wndw    = hanning(numP-1);
```

```
for c=1:numC,
        w(:,c) = wndw;
end;
%
%       doppler filter
%
xx = zeros(numP,numC);
for ir = 1:numR,

        xx(1:(numP-1),:) = w .* x((ir-
1)*numP+1:ir*numP-1,:);
        xx(numP,:)=zeros(1,numC);

        dcr(1:numC*numP,ir) =
reshape(fft(xx),numP*numC,1);

        xx(1:(numP-1),:) = w .* x((ir-
1)*numP+2:ir*numP,:);

        dcr(numC*numP+1 : 2*numC*numP ,ir) =
reshape(fft(xx),numP*numC,1);
end;
```

A13RawToFFT2

```
function [dcr] = rawToFFT2(x, numP, numR, numC)
numRnumC  = numR*numC;
numPnumC  = numP*numC;
numPnumR  = numP*numR;
%
%       range weight data
%

range_offset       = 116;
inv_denom= 1 ./ (32768*range_offset*range_offset);
l                  =
range_offset+floor(linspace(0,(numPnumR-1)./numP,
numPnumR));
l                  = l .* l * inv_denom;
l                  = l.';
rwght              = l * ones(1,numC);
x                  = rwght .* x;
%
%       compute doppler weighting matrix
%

wndw               = hamming(numP-1);
wndwpad            = [wndw ; 0];
dwght              = wndwpad * ones(1, numRnumC);

%
%       reshape data
%

x                  = reshape(x, numP, numRnumC);

%
%       doppler filter
%

xx                 = fft(x .* dwght);

dcr                = zeros(2*numPnumC,numR);

%
%       reorder data
%

for c=0:numC-1
        dcr(c*numP+1:(c+1)*numP,:)   = xx(:,
c*numR+1:(c+1)*numR);
end;

xclear(xx);

%
%       doppler filter
%

x                  = [x(2:numP,:) ;
zeros(1,numRnumC)];
xx                 = fft(x .* dwght);

%
%       reorder data
%

for c=0:numC-1
        dcr(c*numP+1+numPnumC:(c+1)*numP+numPnumC,:)
        = xx(:, c*numR+1:(c+1)*numR);
end;
```

ACKNOWLEDGEMENTS

The authors wish to thank the personnel at DARPA for sponsoring this effort and the personnel at the Maui High Performance Computing Center the allowing them to complete this report. The authors wish to thank Luiz DeRose from University of Illinois, Fred Pearson from Lincoln Laboratory for their donations of test code. Further, the support from Air Force Research Laboratory, Science Applications International Corporation, and Albuquerque High Performance Research Center is also appreciated.

REFERENCES

[1] The MathWorks Inc, MATLAB Compiler User's Guide, 1995.

[2] V. Menon, A. Trefethen, "MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing", SuperComputing '97 Technical Paper.

[3] Real Time Express, http://www.rtexpress.com/.

[4] Luiz De Rose and David Padua, A MATLAB to Fortran 90 Translator and its Effectiveness, 10th ACM International Conference on Supercomputing, May 1996.

[5] The RTExpress distribution example codes.

[6] The Ground Processing Space Time Adaptive Processing (STAP) Suite courtesy of Massachusettes Institute of Technology/ Lincoln Laboratory.

[7] P. L. Springer, Matpar: Parallel Extensions for MATLAB, http://www-hpc.jpl.nasa.gov/PS/MATPAR/index.html.

[8] Joel Hollingsworth, Kun Liu, and Paul Pauca, Parallel Toolbox for MATLAB, Wake Forest University, http://www.mthcfc.wfu.edu/pt/pt.html.

[9] H. Anton, Elementary Linear Algebra, 1984, John Wiley and Sons, p69-86.

15. Using MultiMATLAB at the MHPCC

F. Melody Bohn
Maui Community College (MCC)
D.J. Fabozzi
Maui High Performance Computing Center (MHPCC)

3 February, 1999

To Support Contract Statement of Work Subtask 4.1.4.1, Investigate and implement fine grain parallelization over the MHPCC SP-2 nodes in the Khoros 1.5 environment of the RLSTAP/ADT and MATLAB.

Using MultiMATLAB At The MHPCC

F. Melody Bohn
Maui Community College
kas@maui.net

D.J. Fabozzi
Maui High Performance Computing Center
Kihei, HI, 96753
fabozzi@mphcc.edu

February 3, 1999

Introduction

MultiMATLAB[1] is a utility developed by the Cornell Theory Center (CTC) which allows MATLAB® processes to be distributed across multiple processors. The user instruments native MATLAB code with MultiMATLAB calls which perform the data distribution and process control in a Massively Parallel Processing (MPP) environment. This report details the MultiMATLAB architecture and the steps to performing MultiMATLAB processes at the Maui High Performance Computing Center (MHPCC). This report also contains source listings of necessary support files as well as a reference of all of the commands in the MultiMATLAB library.

The MultiMATLAB Architecture at the MHPCC

The parallel environment at the MHPCC is built on the IBM RS/6000 Scalable POWERParallel (SP) platform operating AIX version 4.2 and a variety of support software for the development and execution of parallel programs. The MHPCC also offers a variety of mathematical and scientific libraries including latest version of MATLAB which, at the time of this writing, is version 5.2. Where appropriate, these libraries are identified in this report. For additional info about the MHPCC, see http://www.mhpcc.edu.

The MultiMATLAB [2] architecture is implemented on the communication substrate MPICH [3], the public domain version of MPI (Message-Passing Interface) developed by Argonne National Laboratory and Mississippi State University allowing portable implementation across a network of UNIX workstations on a parallel platform. The P4 version of MPICH allows interprocessor communication between a network of workstations connected by TCP/IP.

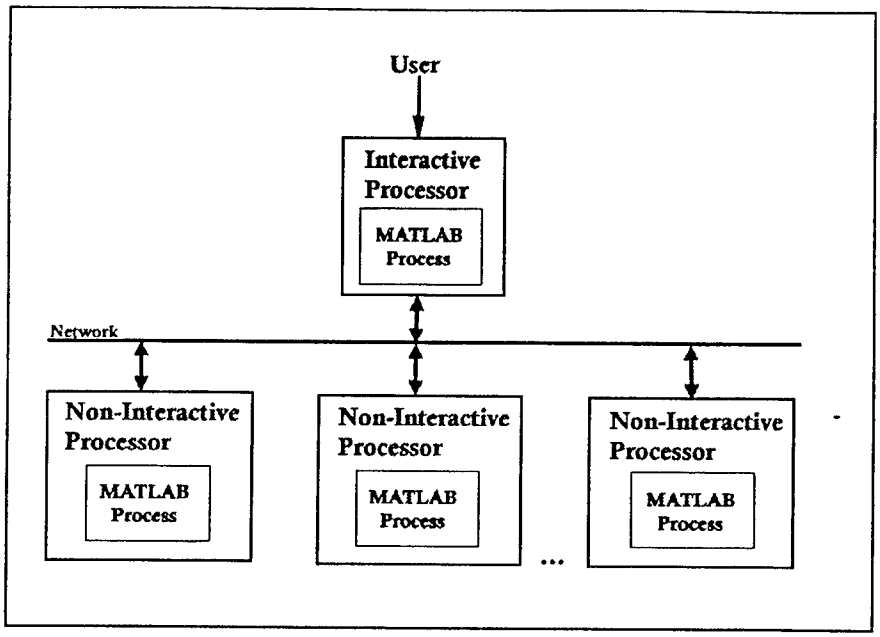


Figure 1 Relationship between distributed processors

To execute an algorithm using MultiMATLAB [4], the user first launches MATLAB on the master processor (interactive processor) which then broadcasts commands to slave processors. Each slave processor launches its own MATLAB process and a MultiMATLAB MEX routine to perform the distributed operation. MEX (MATLAB Executable) [5] [6] routines are external subroutines written in C or Fortran which are executed from the MATLAB environment. The parallel MEX slaves communicate directly with the MATLAB master process via the communication layer.

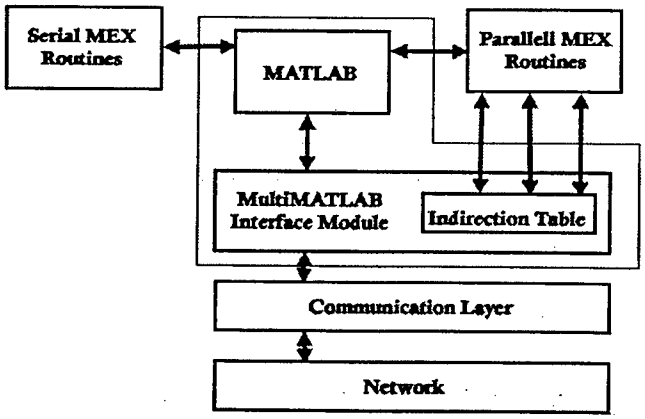


Figure 2 MultiMATLAB Architecture

The MultiMATLAB Interface Module initializes the communication layer and allows interaction between the parallel MEX routines and the communication layer. The interface module resides in each processor's MATLAB address space and allows the parallel MEX routines to access the network without the intervention of the operating system.

When the interface module is first initialized it builds a table of pointers to all functions and data pertaining to the communication layer. Subsequently, when a parallel MEX routine makes a call to MultiMATLAB, the function makes a call to MATLAB only once and passes the location of this table. The routine may now access this table and retrieve any communication layer information it will need using the corresponding table offset.

Executing MultiMATLAB at the MHPCC
The following steps illustrate how to implement MultiMATLAB experiments:

1. **Unix environment set up**
2. **Serial program development**
3. **Parallel program development**
4. **Batch job execution**

1. Unix environment set up
The user's .cshrc or .login file must contain the following lines:

```
setenv MM_HOME [path]
setenv MATLABPATH [path]
setenv PATH .:[path]
```

The actual paths at the time of this writing are:

```
setenv PATH .:/usr/nfs/packages/math/MATLAB5/bin:$PATH
setenv MM_HOME /s/crest/djf/MATLAB/MultiMATLAB/final
setenv MATLABPATH /s/crest/djf/MATLAB/MultiMATLAB/final/interactive
```

2. Serial program development

Develop the serial version using known data and results. Figure 3 illustrates our test code serialMul.m, which performs a matrix multiply. The algorithm creates two matrices A and B of dimensions n^2, populated with random numbers. The i^{th} row of matrix A and the j^{th} column of matrix B are multiplied and the sum-product is returned to the i^{th} row/j^{th} column of matrix C.

```
% serialMul.m
% This MultiMATLAB M-file serial matrix multiplication
%

function z = serialMul (n);

A = randn(n);
```

```
B = randn(n);
tic;
  for i = 1 : n
    for j = 1 : n
      C( i, j ) = A( i, : ) * B( :, j ) ;
    end
  end
z = toc;
```

Figure 3. serialMul.m - serial matrix multiplication

3. Parallel program development

MultiMATLAB allows either functional or data partitioning and this guide will examine the latter. As the examples will show, MultiMATLAB commands, which are distinguished from MATLAB commands by beginning with capital letters, are interleaved directly into the source MATLAB code.

As shown in Figure 4, the original program broadcasts the matrices to the slave processors and calls the new routine, remMul.m, Figure 5, which performs the partitioned matrix multiply. The master processor creates three matrices (A, B, and C) on the master processor. MultiMATLAB commands pass matrices A, B, and C to slave processors which execute each respective slave routine. When the routines are completed on all slave processors, the submatrices in "C" are returned to the master processor

```
% matMul.m
% This MultiMATLAB M-file performs matrix multiplication of two ( n x n ) square matrices
% w/ random entries for scalability timing tests. Assumes n (matrix dimensions) & Nproc (#
% of processors) exist in MATLAB variable  space Returns time of execution

function z = matMul (n)
A = randn(n);
B = randn(n);
C = zeros(n);
tic;
Bcast( 'A' );
Bcast( 'B' );
Bcast( 'C' );

Eval( 'remMul' );
D = Sum( 'C' ) ;
```

Figure 4 Master program broadcast and retrieve data

```
% remMul.m    May 1998
```

```
%
% This MultiMATLAB M-file is called by matMul.m to
% perform matrix multiplication on each processor for
% its own individual slice of the resulting matrix.
%

% disp( 'commencing remote multiplication')

l = ID * n / Nproc + 1 ;
u = (ID+1)* n / Nproc  ;

  for i = 1 : n
    for j = l : u
      C( i, j ) = A( i, : ) * B( :, j ) ;
    end
  end

% C

% disp( 'done multiplying remotely' )
```

Figure 5 Slave processor execution

The matrix multiply is partitioned and performed on each processor according to limits set by values of "l" (lower) and "u" (upper). For example, Figure 6 identifies those bounds placed by "l" and "u" for given values of n, ID, and Nproc:

n = size of matrix = 100
ID = ID of processor to do the processing = assigned during execution [0:Nproc-1]
Nproc = number of processors = 5
l = start column number used in remote routine
u = end column number used in remote routine

ID	l = fix[(ID-1) * n / (Nproc-1) + 1]	u = fix[(ID * n) / (Nproc-1)]
1	1	25
2	26	50
3	51	75
4	76	100

Figure 6 Example data partitioning bounds

4. Batch job execution

Other steps are required for actual execution of MultiMATLAB jobs at the MHPCC. To begin, the master processor needs a list of valid processor names (matMul.hosts) for execution.

This list is created from a call from a loadLeveler script after the job is submitted for execution on the MHPCC system. An example load leveler script, matMul.cmd, is listed in Appendix A. Notice this script invokes the command `bufferStuff`, which performs the dynamic creation of the execution pool node list. Listed in Appendix B, `bufferStuff`, takes the Load leveler hostfile selection and creates a startup.m file containing the valid node names. After which, the `start` command initiates the MultiMATLAB daemon processes on each slave node begins the distributed operation.

Summary

This document detailed the steps of executing MultiMATLAB command files at the Maui High Performance Computing Center. Also included are the necessary command files for creating the target node lists and load leveler command files. As the actual commands may change with any system configuration, see the help@mhpcc.edu for system changes.

Appendix A

Load leveler batch command file: **matMul.cmd**

```
#!/bin/csh
#################################################################################
#FILE: matMul.cmd
#DESCRIPTION:  Matlab Test LoadLeveler command
# USE: llsubmit matMul.cmd
# AUTHOR: 7/1/98 Matt Green    12/98 modified by Melody Bohn
#################################################################################
#####
#@ job_name       = SerialMul

### NOTE: Be sure to substitute your actual userid below
#@ initialdir     = /u/melody/bohn

### If running on MHPCC production system - be sure to remove the "class"
### line below.  This class only applies to the workshop machine.
### class          = Workshop

### For the MHPCC production system, set this to some realistic value for your
### job.
#@ wall_clock_limit  = 36000

#@ output         = $(job_name).$(cluster).$(process).out
#@ error          = $(job_name).$(cluster).$(process).out
#@ job_type       = serial
#@ requirements   = (Adapter == "hps_user")  && (Memory == 128)
#@ min_processors  = 1
#@ environment    = MP_EUILIB=us;MP_INFOLEVEL=1;MP_LABELIO=yes
#@ account_no     = GOVTA-0030-G00
#@ queue

echo $LOADL_PROCESSOR_LIST > matMul.hosts

bufferStuff
matlab;
```

Appendix B

Parallel files: bufferStuff

BufferStuff-creates a startup.m file containing a different valid node names

```
#!/bin/ksh

for X in $(cat matMul.hosts)
do
print $X>>hostList
done

cut -f1 -d'.' hostList >hostList2

n=$(cat hostList2|wc -l);

Z="Start( [ '";

exec 0<hostList2;

while read Y
do
   ((n=n-1))
   if (( 0 < n ))
   then
      if (($(print $Y|wc -c)==7))
      then
         Y="$Y ";
         Z="$Z$Y';'";
      else
         Z="$Z$Y';'";
      fi
   else
      if (($(print $Y|wc -c)==7))
      then
         Y="$Y ";
         Z="$Z$Y' ] )";
      else
         Z="$Z$Y' ] )";
      fi
   fi
done
echo $Z>startup.m

echo "mat_inv4">>startup.m
```

```
echo "Quit">>startup.m
echo "quit">>startup.m

rm hostList
rm hostList2
```

Appendix C

Table of MultiMATLAB Commands

Most commands can be run only on the master process (process 0). ***Commands marked by * can be run on the master process or on the remote processes (1:Nproc-1).***	
Starting and stopping **MultiMATLAB**.	
Start	- Initialize remote processes and begin MultiMATLAB session
Interrupt	- Interrupt MultiMATLAB processes during computation
Abort	- Abort MultiMATLAB session remaining in originally interactive session
Quit	- Terminate remote processes and end MultiMATLAB session
Process arrangement and identification.	
*ID	- Task ID of a process
*Nproc	- Total number of MultiMATLAB processes active
Grid	- Arrange the processes in a grid
*Gridsize	- Dimensions of the grid of processes
*Coord	Coordinates of a process in the grid
Running commands on multiple processes	
Eval	- Evaluate a command on one or more processes
Communication	
*Send	- Send data from one process to another
*Recv	- Receive data sent from another process
*Probe	- Determine if communication has been completed
*Barrier	- Synchronize processes
Put	- Put data from the master process onto remote processes
Get	- Put data from remote process onto the master process
Bcast	- Transmit data to all processes using a tree structure
Distribution	
Distribute	- Distribute a matrix according to the values of Coord
Collect	- Collect a matrix according to the mask created by Distribute
Shift	- Shift data between processes
Arithmetic	
Max	- Find the pointwise maximum of matrices on several processes
Min	- Find the pointwise minimum of matrices on several processes
Sum	- Find the pointwise sum of matrices on several processes
Prod	- Find the pointwise product of matrices on several processes
Graphics	
Window	- Arrange figures in a grid according to function parameters

Reset Window	- Reset default window position to MATLAB default
Refresh	- Repaint all current figures

Appendix E

MultiMATLAB M-Files

Abort *Syntax:* Abort	Abort MultiMATLAB
Barrier *Syntax:* Barrier(a)	Synchronize all processes. The command Barrier returns only when called on each process.
Bcast *Syntax:* Bcast(val,bcast_root,bcast_communicator)	Bcast can be run only on the master process. Broadcast a matrix from one process to all other processes using a tree-structured algorithm. Bcast('A',i, comm) takes process i as the root of the tree and broadcasts A to processes using communicator comm. Comm is an optional argument. root is an optional argument, if it is not defined the root is assumed to be process 0. See also Send, Recv, Put, Get.
Collect *Syntax:* Collect(name)	Pre: ul - the upper left corner of the matrix that was distributed chunk - the string, name of the chunk on each of the processes Post: return the matrix put back together (the opposite from Distribute)
Coord *Syntax:* [i,j,k]=Coord	Coordinates in the grid. [i,j,k]=Coord returns the coordinates of the calling process in the grid set up by Grid. See also Gridsize, Distribute, Collect, Shift, Window.
Distribute *Syntax:* Distribute (name)	Pre: m-by-n matrix name - a string for the name of the matrix chunk. Post: split the matrix into more or less even chunks across the processes in the grid. Split the matrix row-wise first across the first coordinate, then column-wise across the second

	coordinate, and column-wise again across the last coordinate. All the results go into the name value, the master process gets the upper left corner in the return value. Note: the matrix dimensions have to be greater than the grid
Eval *Syntax:* Eval('cmd') *Syntax:* Eval(i, 'cmd ')	Eval can be run only on the master process. Execute commands on one or more processes. Eval('cmd') evaluates the command cmd or the file cmd.m on all processes. Eval(i,'cmd') evaluates cmd or the file cmd.m on process(es) i, where i can be a scalar or a vector.
Get *Syntax:* A=Get(i,Aname)	Get can be run only on the master process. Retrieve data from a remote process and place it on the master process. A=Get(i,'B') gets matrix B from i and puts the data in matrix A. See also Put.
Grid *Syntax:* Grid(m,n) *Syntax:* Grid(m,n,p)	Grid can be run only on the master process. Arrange the processes in a grid. Grid(m,n,p) makes an m x n x p grid, Grid(m,n) makes an m x n x 1 grid. Any process can then obtain the coordinates of the grid with the commands Gridsize and Coord, and the same information is also used by Distribute, Collect, Shift, and Window. When MultiMATLAB is started, the processes are in the arrangement corresponding to Grid(Nproc,1,1).
Gridsize *Syntax:* [i,j,k]=Gridsize	Dimensions of the grid of processes. [m,n,p]=Gridsize returns the dimensions of the grid of processes set up by Grid.

	See also Coord, Distribute, Collect, Shift, Window.
ID *Syntax:* a = ID	Task ID of the process, an integer from 0 to Nproc-1. See also Nproc.
Interrupt *Syntax:* Interrupt(m)	Interrupt a MultiMATLAB process during computation, to return it to a listening state. Input parameter m defines which process should receive an interrupt, if no parameter is provided, all processes receive an interrupt.
Max *Syntax:* res=Max(val)	Max can be run only on the master process. Max('x') is the elementwise maximum of x over all processes. The variable x must exist and have the same dimensions on all processes. See also Min, Sum, Prod.
Min *Syntax:* res=Min(val)	Min can be run only on the master process. Min('x') is the elementwise minimum of x over all processes. The variable x must exist and have the same dimensions on all processes. See also Max, Sum, Prod.
Nproc *Syntax:* a = Nproc	Nproc is the only MultiMATLAB command, aside from Start and Demo that can be issued before MultiMATLAB has been initialized. If MultiMATLAB has not been initialized, the value of Nproc is 0. Total number of MultiMATLAB processes active. See also ID.
Probe	Flag indicating whether a message has arrived.

Syntax: [a,e,f] = Probe(b,c,d)	Probe(i) returns 1 if the message from process i arrived and 0 if it has not. Probe returns 1 if a message from any process has arrived and 0 if no messages have arrived. [f,i] = Probe returns the flag in f and the source of the message in i.
Prod *Syntax:* res=Prod(val)	Prod can be run only on the master process. Prod('x') is the elementwise product of x over all processes. The variable x must exist and have the same dimensions on all processes. See also Max, Min, Sum.
Put *Syntax:* Put('A') *Syntax:* Put(i,'A')	Put can be run only on the master process. Put data from the master process to remote processes. Put(i,'A') places matrix A on process(es) i, where i can be a scalar or a vector. Put('A') places A on all processes. See also Get.
Quit *Syntax:* Quit	Quit can be run only on the master process. Terminate remote MultiMATLAB processes and end MultiMATLAB session, leaving user within standard MATLAB. If MultiMATLAB is running and the standard MATLAB command quit is issued, the MultiMATLAB session is first terminated and then MATLAB is terminated as usual.
Recv [a1,a2,a3] = Recv(b,c,d)	Receive data sent from another process. A=Recv receives any matrix sent to the process and places it in A. A=Recv(i) receives a matrix from process i and places it in A.

	[A,j]=Recv receives any matrix sent to the process and returns the sender's ID in j. [A,j,tag]=Recv returns also the tag of a message. See also Send, Put, Get.
Refresh Refresh(ids)	Repaint all current figures
ResetWindow	Can be called only by the master process, process 0. Resets the position for figure windows to the MATLAB default. See also Window.
Send Send(a,b,c,d)	Send data from one process to another. Send(i,A) sends matrix A to process(es) i, where i can be a scalar or a vector. Optional arguments may be used to indicate a communicator or tag for a message Send(i,A,comm,tag) See also Recv, Put, Get.
Shift Shift (mat, axis, amount, kind)	Pre: A - the matrix chunk on the master process mat - string, the name of the matrix in all processes axis - 1 or 2 amount - the number of rows/cols to be shifted kind - string, 'cshift' circular, 'eoshift' end off shift Post: matrix mat is a distributed matrix (using Distribute) according to the Grid. This function shifts the distributed matrix the processes according to the axis, amount, and the

	kind of the shift. Returns the upper left corner that the master matrix is supposed to have. Therefore, for a distributed matrix A, the call should be something like: A = Shift(A, 'A', 1, 2) Note that omitting the last parameter defaults to a circular shift.
Start Start(n) Start(rhosts)	It is not possible to add new processes after MultiMATLAB is already running. Instead one must Quit and then Start again. Begin a MultiMATLAB session. Start(n) initiates n remote MultiMATLAB processes, bringing the total to n+1 including the master process. The host names are taken from the file $MM_HOME/etc/hostfile. Start(['hostname1';...;'hostnameN']) opens remote MultiMATLAB processes for the hostnames indicated. Hostnames must be the same length in this hostname matrix in order for it to be a valid matrix. Therefore, names should be padded with spaces at the end manually or by using the MATLAB command str2mat, e.g., Start(str2mat('parus','purple','emma')).\
Sum res=Sum(val)	Sum can be run only on the master process. Sum('x') is the elementwise sum of x over all processes. The variable x must exist and have the same dimensions on all processes. See also Max, Min, Prod.
Window Window(m,n)	Set up the position for figure windows to align plots by different processes. Window(M,N) sets up a M by N grid of figures. See also ResetWindow.

References

[1.] V. Menon and A. E. Trefethen
MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing.
Cornell Theory Center
http://simon.cs.cornell.edu/Info/People/vsm/papers/sc97/

[2.] A. E. Trefethen, V. S. Menon, C. C. Chang, G. J. Czajkowski, C. Myers and L. N. Trefethen
MultiMATLAB: MATLAB on Multiple Processors.
Technical Report pp 96-239, Cornell Theory Center, 1996.
http://www.cs.cornell.edu/Info/People/lnt/multimatlab.html

[3.] MPICH-A Portable Implementation of MPI
http://www.mcs.anl.gov/mpi/mpich/

[4.] C. C. Chang, G. J. Czajkowski, X. Liu, V. S. Menon, C. Myers A. E. Trefethen, and L. N. Trefethen
The Cornell MultiMATLAB Project. Cornell Theory Center
http://www.tc.cornell.edu/Software/MultiMATLAB/

[5.] The MathWorks Technical Papers.
http://www.mathworks.com/support/tech-notes/v5/1600/1615.shtml

[6.] The Math Works Inc.
MATLAB User's Guide: Reference Guide. The Math Works Inc. 1992.

16. Distributed Algorithm Stream (DAS)

Mike Koligman
Par Government Systems

4 February, 1999

To Support Contract Statement of Work Subtask 4.1.2.3, Establish and Implement the Airborne Infrared Measurement System (AIRMS) Database.

Distributed Algorithm Stream (DAS)

Maui High Performance Computer Center (MHPCC)

RUNNING DAS

Execution of the DAS follows the procedures outlined in the DAS 1.3 User's and installation manual which can be accessed over the network. This manual (file: install.pdf) is in PDF format and requires a PDF capability such as 'acroread' at the user site. In order to make expeditious use of the DAS, the user must have an understanding of the theory and operation of the various modules.

To execute DAS, the user must:

1. Log onto the SP2 (telnet tsunami.spw.mhpcc.edu) using telnet
   (users must be registered with MHPCC and have an existing account)
   reference http://www.mhpcc.edu.com for account access information.

2. Source the `$BASCOM/das.csh` setup file after login.

3. Verify that the link to the $PVM_ROOT directory in the user's home directory:
   (e.g. cd; ln -s $PVM_ROOT)

4. The .rhosts file in the users home directory must contain the list of nodes as well. A sample file is located in $BASETC/rs6000/rhosts which can be copied to ~/.rhosts. The permission on this copied file must be set via 'chmod 600 ~/.rhosts')

5. Be in possession of the necessary namelists and data.

The various modules can be run serially by executing the modules individually (e.g. typing sa_pn at the command prompt executes the pattern noise module). This is the simplest mode of operation but also the slowest as only one processor is used.

The DAS can also be run across several nodes on the SP2 as a parallel processing environment. This is done using the 'das' executable which can be run in two modes. In the first, the DAS job is executed interactively using the 18 interactive nodes on the SP2. Operation in this manner requires that the user first start the pvm deamons on the various nodes. This is accomplished by supplying a list of nodes to the pvm console:

```
pvm < $BASETC/rs6000/addhosts
```

The file $BASETC/rs6000/addmhosts is a sample list of nodes. The DAS can then be executed interactively as a pvm job. (The user is referred to the documentation accompanying pvm for further information regarding pvm.)

DAS jobs can also be submitted to the MHPCC load-leveler queue using a script similar to the following:

```
#!/bin/csh
#
# Load leveler parameters
#
#@ job_name          = das6
#@ output            = $(job_name) .$(cluster) .$(process) .out
#@ error             = $(job_name) .$(cluster) .$(process) .err
```

```
#@ notification       = always
#@ notify_user       = YOUR_E-MAIL_ADDRESS
#@ checkpoint        = no
#@ job_type          = pvm3
#@ parallel_path     = /u/joea/pvm3/bin/RS6K
#@ requirements      = (Adapter =="hps_ip")
#@ min_processors    = 6
#@ max_processors    = 6
#@ wall_clock_limit  = 5:0:0
#@ environment       =MP_INFOLEVEL=5
#@ queue

# Run das

echo RUNNING das.cmd
cd $BASSRC/tst
das
echo Execution Complete
```

When using this script, you need to specify the correct number of nodes (6 above) and your e-mail address.

IMPORTANT: THIS SCRIPT IS *NOT* RUN INTERACTIVELY BUT IS SUBMITTED TO THE LOAD-LEVELER QUEUE USING:

```
llsubmit das.cmd
```

Man pages for the load-leveler are available at the MHPCC.

INSTALLING DAS

The MHPCC version of DAS is based on DAS V.1.3 which was the final version released under the AIRMS program. DAS 1.3 was primarily written Fortran 77 and runs on a variety of platforms.

The SP2 environment at the MHPCC supports Fortran 90. Though the two languages are similar, there are enough differences between them that numberous code modification were required for porting DAS. These generally relate to syntax differences between modules which do not affect the operation of the algorithms.

The bulk of the code translations are made automatically by a preprocessor program developed under this effort. This program, preproc, takes in the original F77 code, recognizes the various syntactical differences, and generates a F90 source file that can be compiled by the IBM compiler. In a few instances, certain F77 codes had to be changed manually to accommodate both F77 and F90 with the same source file.

To preserve compatibility with the prior machines and the SP2, conditional compilation (i.e. IF DEFS) constructs were inserted into the code.

The entire DAS package represents approximately 500k lines of code. Only a subset of the DAS algorithms were ported to the SP2. This set of modules includes the "baseline" stream that was used under the AIRMS program. Hence, the SP2 user has access to a very rich, powerful, and well-tested 3-D signal processing stream.

The modules that were ported include:

AP_CAL	– Calibration
SA_SB	– Sticky-bit removal

AP_TI	– Additive target injector
AP_TO	– Occulting target injector
SA_PN	– Pattern noise mitigation
RS_RS	– Registration
RS_SG	– Scene segmentor
FL_IR	– Covariance-based 3-D filter
FL_AM	– 2-D anti-median filter
RS_RS_ST	– Reverse registration
FL_SP_VD	– Velocity stacker
DT_BN	– CFAR detection & grouping
UT_2T_FD	– Frame difference
AP_DR	– Data rework
DAS	– Parallel executable of the entire algorithm stream

In addition, the following utilities have also been ported:

ARGTOOL	– Command line argument pre-processor
AP_GET_CAL_COEFFS	– Retrieves cal coefficients from raw data
PREPROC	– F77 to F90 converter
UT_CJPEG	– Creates jpeg-like images from a BAS file
UT_COMBINE_JPEG	– Combines DAS jpeg files
UT_DJPEG	– Decodes a DAS jpeg file

NOTE: The file format used in the DAS jpeg imagery is NOT compatible with the industry standard format. (The above tools can, however, be used to manipulate these files.)

Descriptions and operations of the various modules can be found in the on-line documentation.

INSTALLATION PROCEDURE

The DAS installation at the MHPCC follows the general procedure outlined in the DAS User's and Installation manual which is on-line (file: install.pdf) and can be accessed over the network. This manual is in PDF format and requires a PDF capability such as 'acroread' at the user site.

The installation procedure assumes that the DAS directory structure is loaded into a directory $BASROOT and that the user's default shell is CSH or TCSH:

1. Edit the file $BASROOT/com/unix/das.csh and change the line

   setenv BASROOT/.....

   to point to the current BASROOT directory.

2. Source this file to set up the DAS environment.

3. cd $BASSRC

4. bld_rs6000

Compilation errors that are reported can generally be ignored since they refer to portions of the code that were not part of the baseline SP2 port. Of course, the user must have the necessary permission to rebuild the package.

SAMPLE DATA

A sample run was created in $BASTST/rs6000 which can be used for familiarization and verification purposes. The DAS installation can be checked using this data:

```
1.  cd $BASTST/rs6000
2.  md test
3.  cd test
4.  cp ../[A-Z]* ../parent.nml  ../test.irst
5.  das
6.  Use optimage on one of the suns at MHPCC to view the
    results and compare to those in $BASTST/rs6000
```

REQUIRED NAMELIST PROCESSING PARAMETERS

1. (AP_CAL_PARAM) - Specify Calibration Coefficients from the following list:

CAL_OFFSET – Offset
CAL_SLOPE – Gain

Flight Number	Filter Number		Gain	Offset
13	4		3.4716167e-04	-6.23364
14	4		3.5e-04	-6.5
15	4		2.54e-04	-1.1 (old)
15	4		3.18e-04	-5.91
16	1		2.587e-03	-79.29
16	3		bad data	bad data
16	4		3.546e-04	-7.004
16	6		1.596e-02	-514.79
19	4		3.556813e-04	-5.60775
20	4		3.306e-4	-6.285
22	4		3.29e-04	-4.6
23	4		3.41e-04	-5.08
26	4	(TDI ON)	3.31286e-4	-4.681
26	4	(TDI OFF)	1.358e-3	-37.935
28	4		3.09516e-4	-2.661
29	4	(TDI OFF)	1.331e-3	-37.0858
29	4	(TDI ON)	3.09516e-4	-2.661279
31	4		3.3567e-4	-4.27129
32	4	(TDI ON)	3.35826e-4	-4.215411
33	4	(TDI ON)	none available used coefficients	from flight 29's
33	4	(TDI OFF)	1.3455e-3	-37.8335
34	4		3.3642424e-4	-4.820657
35	1		1.2789777e-3	-35.51252
	2		5.1267230e-4	-9.811104
	4		3.3298979e-4	-4.174171
39	4		3.3998e-4	-4.4687
40	4		3.50363e-4	-6.44521
41	4		2.77728e-4	-3.84992
42	4		2.2491126e-4	-1.989383
45	4		3.43381e-4	-4.85285
46	4		3.7540126e-4	-6.759626
47	1		1.28845e-3	-34.4770

17. MHPCC Web-Based IR Data Library Simulation

Mike Koligman
Joe Attili
Par Government Systems
David Quinn-Jacobs
ReQuest Technologies

November, 1998

To Support Contract Statement of Work Subtask 4.1.2.3, Establish and Implement the Airborne Infrared Measurement System (AIRMS) Database.

MHPCC Web-based IR Data Library and Simulation

Michael Koligman[1], David Quinn-Jacobs[2], and Joseph B. Attili[3]

PAR Government Systems Corporation
San Diego Technology Center
1010 Prospect Street, Suite 200
La Jolla, CA 92037

ABSTRACT

The Maui High Performance Computing Center (MHPCC)[4] has recently established a remote IR Data Library and Web-based Simulation environment. This environment enables a large number of researchers to access, review, and process IR data sets via the internet. The IR data was collected with the ARPA-sponsored Airborne IR Measurement System (AIRMS) sensor under the completed AIRMS program[1].

This paper describes a cost-effective approach for providing internet access to the AIRMS data library and pre-processing algorithms. The purpose of the effort was to make the data collected with AIRMS and related data documentation easily accessible via the Internet by researchers from a wide community of qualified organizations including government laboratories, universities, contractors, and other interested parties. In addition, users will have the ability to view and process selected data segments stored at MHPCC. After pre-processing, users can transfer both raw and processed data segments to their local facilities. The pre-processing algorithms, which were designed and implemented under the AIRMS program consist of data cleanup, data rework and other signal pre-processing functions. This capability also provided a test bed for future signal processing algorithm implementations.

Keywords: IR Data, AIRMS, IRST Simulation, IR Processing, IR Signal Processing

1. INTRODUCTION

The infrared (IR) data collected by ARPA's AIRMS sensor contains a wide range of backgrounds and targets which can be used by researchers for many different applications. The goal of the MHPCC Web-based effort was to provide AIRMS data to the research and development community in a user-friendly environment which was easily accessible over the internet. The data library and associated software resides at the Maui High Performance Computing Center.

Various research and government agencies are currently investing in or considering IR sensors as part of a multispectral-multisensor approach to high probability of detection / low false alarm rate detection, identification, and tracking of targets in clutter. Applications of interest include:

- Theater surveillance
- Fleet self-defense
- Target search and track
- Aircraft missile threat warning
- Ship missile threat warning / targeting
- Missile seekers
- Theater ballistic missile defense

Research is ongoing into both the signal/track processing algorithms and the associated IR phenomenology. IR data which can support this research is not readily available from traditional sources and collection programs. The AIRMS program has amassed a large body of high quality, high resolution IR imagery which currently resides at both the Naval Air Warfare

[1] M.K.: Email: Mike_Koligman@PARtech.com Telephone: (619) 551-9880; Fax: (619) 551-0257
[2] DQJ.: Email: dqj@RequestTech.com Telephone: (607) 275-3000
[3] J.A.: Email: Joea@PARtech.com Telephone: (619) 551-9880
[4] The work described in this paper was performed by MHPCC, PAR Government Systems Corp., and Request Technologies

Center-Weapons Division (NAWC-WPNS, China Lake, CA) and the MHPCC. This data can be very useful to system designers, algorithm developers, and physicists for addressing a wide range of research problems and issues.
The AIRMS signal processing stream was originally developed and implemented on an 80 node Intel Paragon parallel processor which was also used to perform the data processing[2]. The algorithms and data have since been ported to an IBM SP2 parallel processor located at the MHPCC. The algorithm stream, Distributed Algorithm Stream (DAS), is representative of current state-of-the-art processing techniques, and as such, can serve as a performance baseline for algorithm research.

Figure 1 shows the AIRMS aircraft and associated MW/LWIR sensor which was used to collect the data. The aircraft was flown on over 50 missions with a total of over 1 Terabyte of data collected. Many different types of sky and terrain backgrounds were collected including clouds, desert, mountains, farmland and ocean. In addition, a variety of targets were collected including various aircraft, cruise missiles, submarines and ground targets.

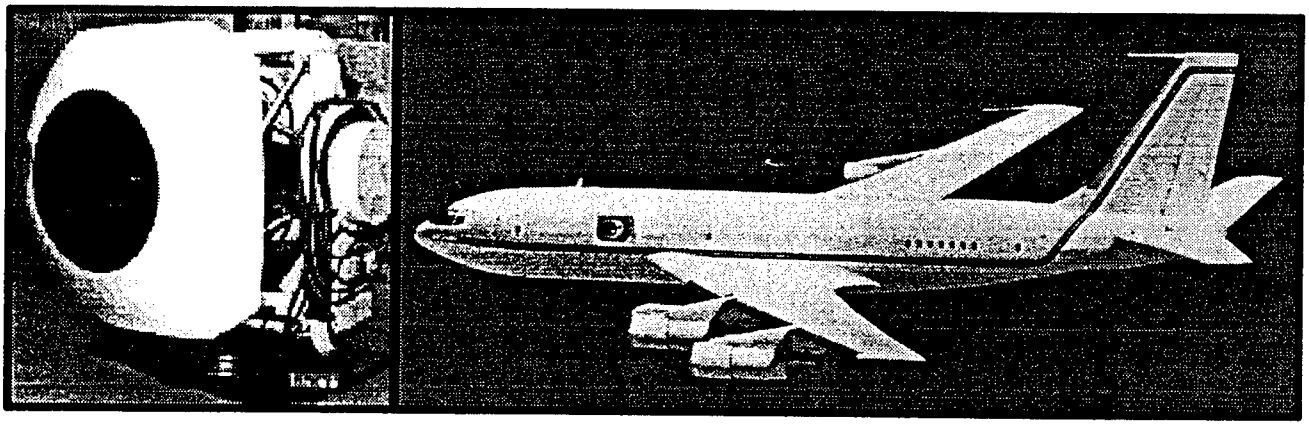


Figure 1. AIRMS 720B Aircraft with 24" Aperture MW/LWIR Sensor

The ability to access AIRMS data, pre-processing tools, and documentation over the Internet represents a new paradigm for disseminating data. It exploits investments in the Internet and the MHPCC to provide much broader access to the data than is possible with the traditional US mail-based distribution of overview documents, sample imagery, and data tapes that have been employed by other IR data collection efforts, such as the IRAMMP and IRMS programs.

1.1 Web-based Data Library Benefits

The Web-based data library offers many benefits to the IR research and development community. Never before has data of such high quality been available as there is now with the AIRMS data. Nor has there been data collected from a single sensor over such a wide range of backgrounds, targets, atmospheric conditions, and wavebands. The AIRMS data is unique because of the extreme high resolution of the sensor. Several key sensor attributes are:

- a very high spatial (8 μrad pixel pitch) and temporal (8.7 Hz frame rate) resolution;
- covers a very broad spectral range from 3 to 16 μm by virtue of its helium cooled, extrinsic silicon detector array;
- very low jitter (on the order of 4 - 6 μrad);
- high sensitivity with low system and residual pattern noise levels (under 100 mK in the 8.2 to 12.2 μm band);
- internal thermal reference sources for accurate radiometric calibration;
- accurate determination of its position and the position of target aircraft it observes through use of GPS receivers on both the AIRMS platform and the target aircraft.

The AIRMS data and pre-processing algorithm stream will provide benefits for many years to come. The potential uses of the data are many including developing new algorithm streams, evaluating algorithm streams developed for new sensors, and mapping AIRMS data to user specified sensor parameters (data rework). Historically, there has been a scarcity of high quality data available during the algorithm development phase of new IR systems. Prior to the AIRMS program, most data came from the Navy IRAMMP sensor or the Air Force/Lincoln Laboratory IRMS sensor, both of which have deficiencies which limit their utility (*e.g.*, aliasing, residual sensor artifacts). The available data often did not include background, target,

or engagement geometry's similar to the intended application of the new sensor. Historically, algorithms have been evaluated on very limited sets of data with questionable results. Running algorithm sets on a standard AIRMS data test set can provide very useful metrics on the overall performance of proposed algorithms. In fact, much of the AIRMS data was processed and analyzed for target detection performance in the presence of clutter. Processed results can be referenced in the available on-line documentation. The AIRMS data can truly be used in levying testing requirements on the developers of new algorithms and provide increased understanding of and confidence in the interpretation of the results.

2. REMOTE ACCESS IMPLEMENTATION

The basic implementation for the AIRMS Remote Data Library and Processing Environment is to have researchers working at their own facilities on their own workstations, but actually using resources located at the MHPCC. Figure 2 illustrates the technical concept. The user has simultaneous access to all the assets of his or her local environment (*e.g.*, custom algorithm software, MATLAB, *etc.*), the AIRMS Data and Data Librarian, the AIRMS pre-processing algorithms and general purpose image display functions.

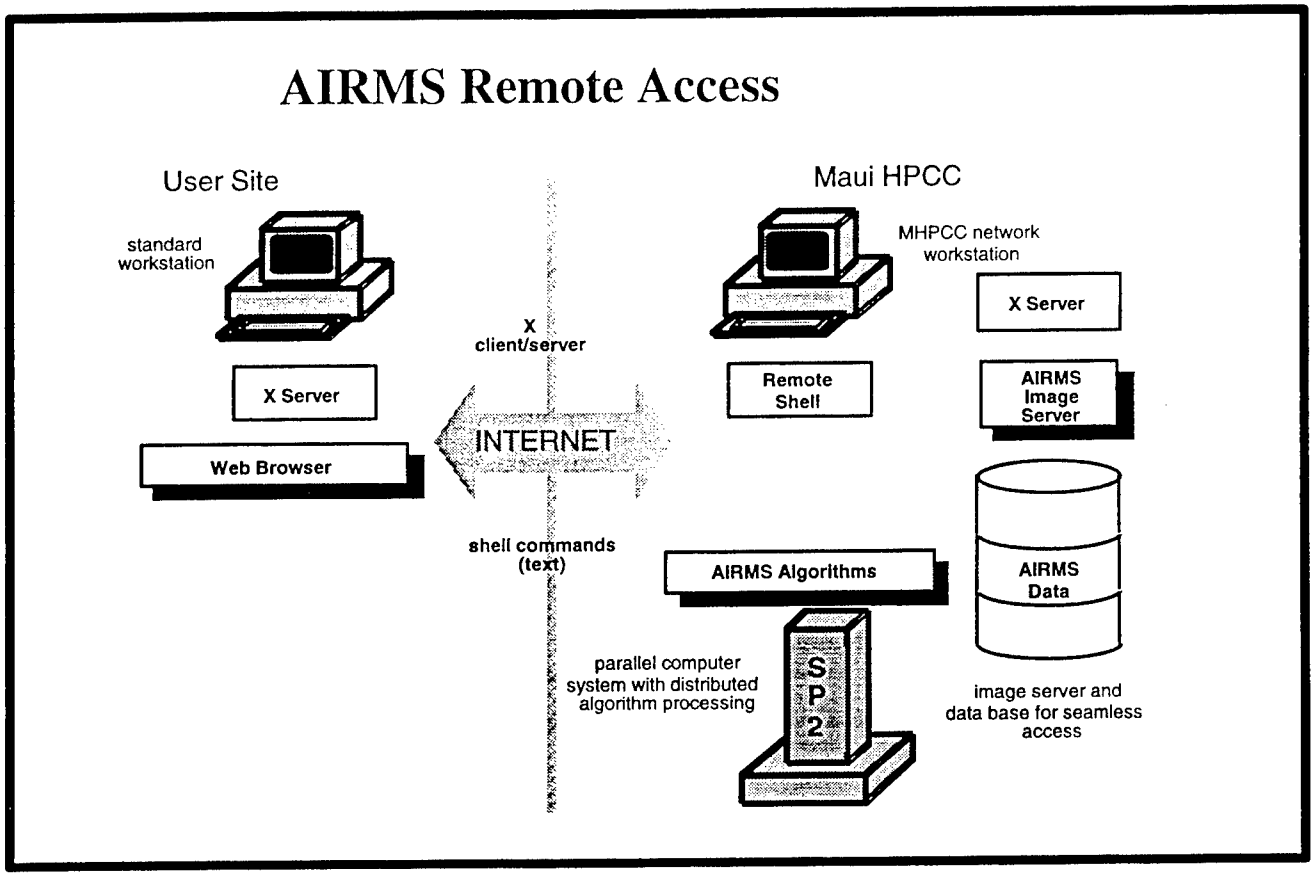


Figure 2. AIRMS Data Library and Processing Environment

To make the data collected with AIRMS easily accessible, a simple data librarian is available to perform search and retrieval functions over the internet. The pre-processing algorithms, which can be run on the IBM SP2 parallel computer located at the MHPCC, are also easily accessed. These pre-processing algorithms allow users to tailor AIRMS raw data to a particular application. The data search capability requires little or no prior knowledge of AIRMS. Most users will be able to perform simple searches immediately upon entering the web site. An on-line data storage capability is also available as this is where the actual AIRMS data is stored. By making the MHPCC parallel computing system available to researchers, processing time can be significantly reduced and research productivity similarly enhanced.

Figure 3 shows the IBM SP2 parallel computer located at the MHPCC. The SP2 is responsible for executing all of the AIRMS pre-processing algorithms and associated signal processing tools. The SP2 contains over 500 processing nodes with

166 GFLOPS of processing power and 82 GB of memory. The MHPCC also has a 25 TB tape server storage capacity which is used to store the on-line AIRMS data. A Sun SparcStation, also located at MHPCC, is used as the server and contains all web interface software. In addition, MPEG imagery and a vast amount of AIRMS documentation is stored on the workstation. Documentation consists of software reference manuals, bulk data processing reports, and other useful reference materials.

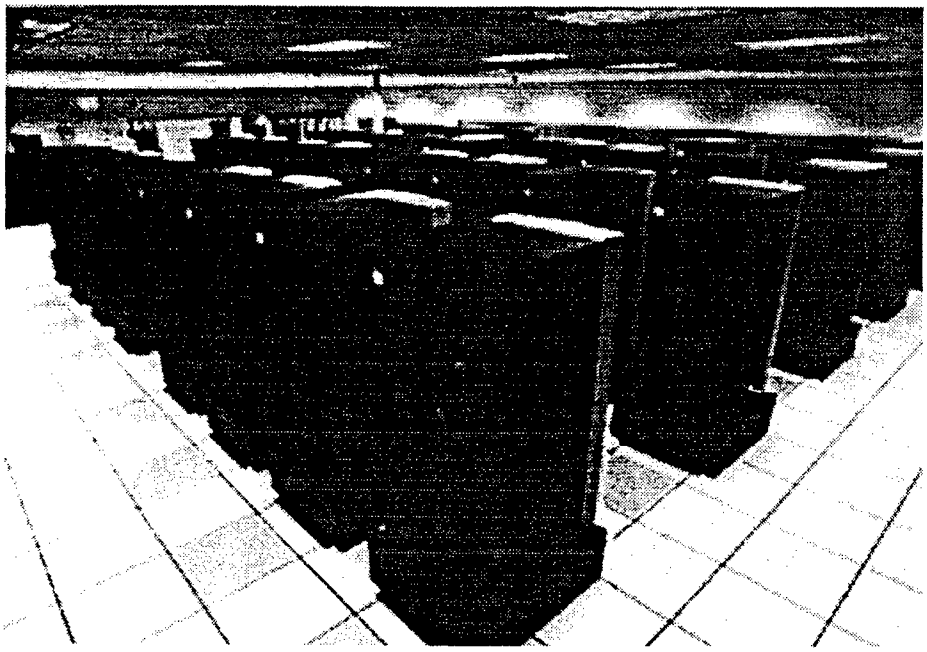


Figure 3. IBM SP-2 - RS-6000 Unix Processors. Peak throughput = 166 GFLOPS

2.1 Web-based Implementation

The AIRMS Data Library and Processing Environment is comprised of the data librarian, on-line AIRMS data base, processing tools, general purpose display and analysis tools.

An architecture was needed in which these elements could be integrated to form a cohesive, easy-to-use system. Figure 2 showed the AIRMS Data Library and Processing Environment software architecture. Software components are located either at the user's site (the left hand side of the figure) or at the MHPCC (the right hand side of the figure). Communication between the two sites is performed over the Internet, but transparent to the user. (Either a WWW or an X client-server can be used; the user merely sees multiple windows on the workstation screen).

The AIRMS Data Librarian is resident at and runs on a Sun SparcStation. The AIRMS Data Librarian can be used either through a WWW browser or directly by logging in to the MHPCC. The AIRMS data files and the data catalog are also resident at the MHPCC. AIRMS data can be downloaded to the user's site or a reference to the data can be saved for later use by an AIRMS process. During the execution of the process, the reference is resolved and the actual data is retrieved.

The AIRMS pre-processing software is also resident at the MHPCC and, at the user's discretion, also at the user's site (approved user's can download the software using a standard FTP service). This software provides the capability of processing data with the AIRMS algorithm stream distributed among designated SP2 nodes. Runs can be prepared without logging in to the MHPCC (all set-up information for using the pre-processing algorithms is input using standard FORTRAN NAMELIST files). When the user is ready to initiate processing, he or she will log in to the MHPCC and initiate the processing using the NAMELIST files previously created.

The Data Librarian provides an interactive data query and retrieval utility. This utility searches the data files based on user defined selection criteria. The actual IR data is comprised of hundreds of files, therefore file descriptors (*e.g.*, test ID, date, time, location, *etc.*) are extracted from each data file and stored in a data catalog file. It is this catalog file which is actually searched. The user's interface to the librarian is through a WWW browser and associated query form. (The librarian is

actually an extension to the MHPCC WWW server implemented using the Common Gateway Interface, the standard method of extending WWW servers.)

The query form provides the means to specify a data base query statement without knowledge of a query language such as SQL. Queries are combinations of the various descriptors with logical operators (*e.g.*, all flights after April 1995 over urban areas at night containing an F-4 target). After submitting a query, the query results (*i.e.*, a list of all data files which satisfied the query criteria) are displayed. The complete set of descriptors associated with a file can be displayed by selecting the file name in the list.

To support processing at a user site, the librarian provides a means to download the selected data. Users must be registered at the MHPCC before downloading data or accessing controlled database parameters. (When a download is requested, the user must provide a password to enable the transfer to proceed; only authorized users are permitted access to AIRMS data). If processing at the MHPCC is desired, the data selections are stored in a text file which is later accessed by the pre-processing software. (the Data Librarian provides a direct interface with the user in a client-server-server architecture). AIRMS data can then be displayed and visually evaluated when desired.

3. ACCESSING AND PROCESSING AIRMS DATA

The ability to access AIRMS data and pre-processing algorithms is performed via the internet through a web browser such as Netscape or the Microsoft Internet Explorer. The AIRMS site will be available for access in the spring of 1998. Interested users can contact the MHPCC or search the internet on any of the keywords provided in the abstract portion of this paper to determine the site address. Upon entering the AIRMS site, a list of flight data (left panel on browser window) is presented including the options to select a simple or advanced queries. The simple query window allows searching basic parameter choices from the AIRMS database. The advanced query allows specifying additional parameter keywords in addition to the basic ones contained in the simple query. The advanced query requires some familiarity with the AIRMS database while the simple query does not.

Figure 4 shows an simple query screen as viewed over the internet. Any number of search parameters can be selected (all possible choices are presented when appropriate) in addition to specifying logic for the search (i.e. equals, not equals, greater than, ect.). After making the search selections, users will submit the search by clicking on the submit button. Any matching AIRMS flight data will then be presented in the left panel. The database contents will then be displayed for the matching data. An MPEG image of the data will also be presented for previewing selected data.

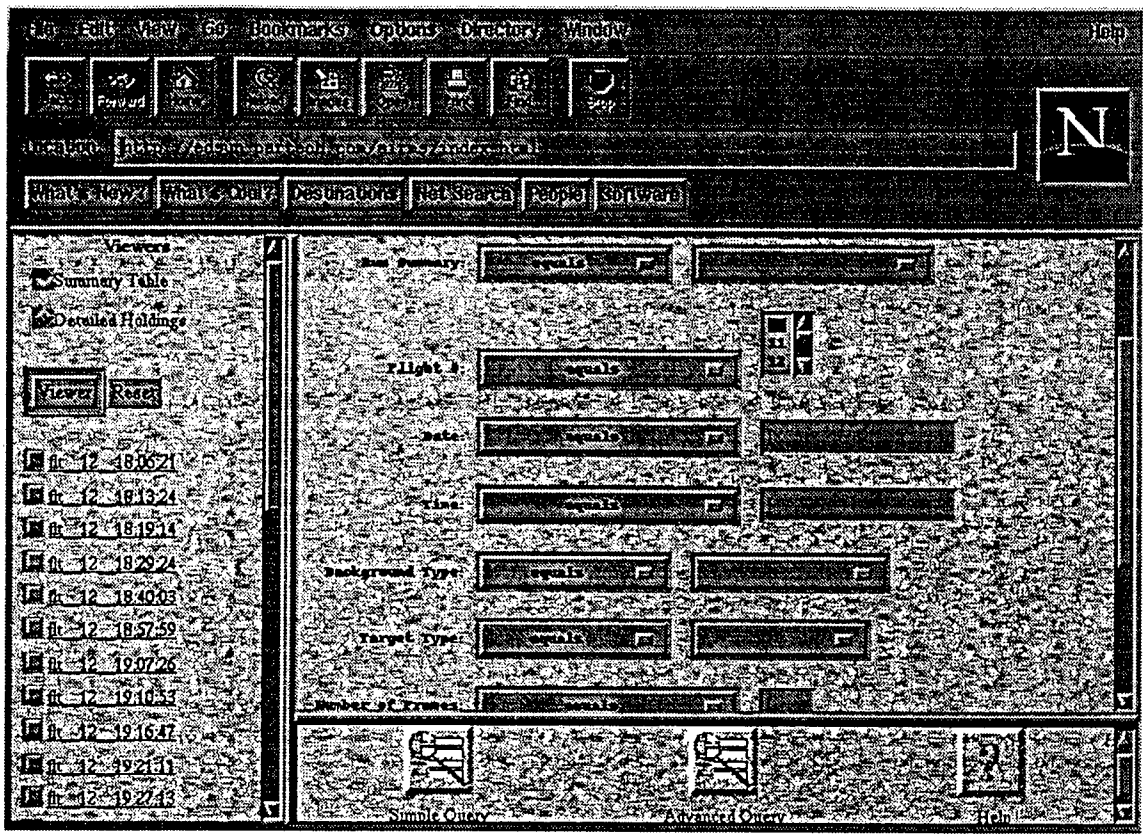


Figure 4. Simple Query Window

Many of the search parameters include a list of choices which can be selected. Two of the most useful search choices are *Background Type* and *Target Type*. Figure 5 shows the possible search choices for each of these two categories. The user can simply select a background and/or target type for the search. All matching AIRMS data flights containing the selected choices will then be presented in the left panel of the browser window. The search engine allows searching on all of the available parameters or selectively specify as many as required to narrow a search. For example, a user may wish to find all cloud data with a G-II target in the background collected only in December. For this example, the user would select *Clouds* using the background type option button and select *G-II* from target type list. Finally, the user would specify *contains* and *December* using the Date option. All matching flight data would then be returned in the left panel. From here, detailed flight information can be displayed for any or all of the returned matching flights. A separate help facility describing search options is available by clicking the help button located at the lower right screen of the browser.

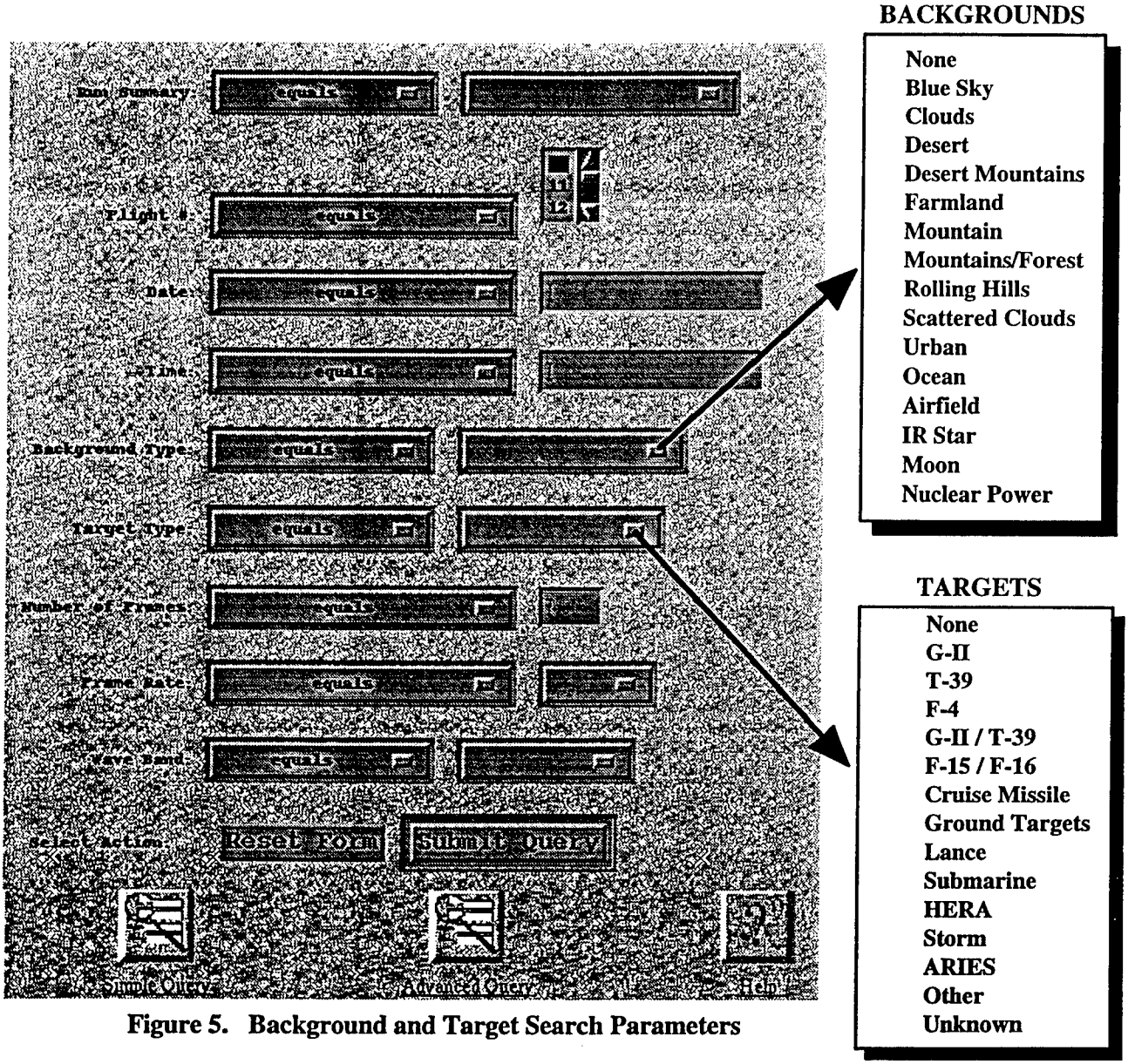


Figure 5. Background and Target Search Parameters

One or more flights can be selected to examine detailed database information. Figure 6 shows a sample detailed output for Flight 12. A sample MPEG image is also display for the first frame of data. If desired, all available imagery frames can be displayed in a movie type fashion. Available documentation can be viewed for selected flights by clicking either the Flight Report, Final Tech Report or documentation buttons. Several fields from the data base can accessed only by registered users. The *secure data* allows users to supply the MHPCC server with a password at which time restricted data can be viewed.

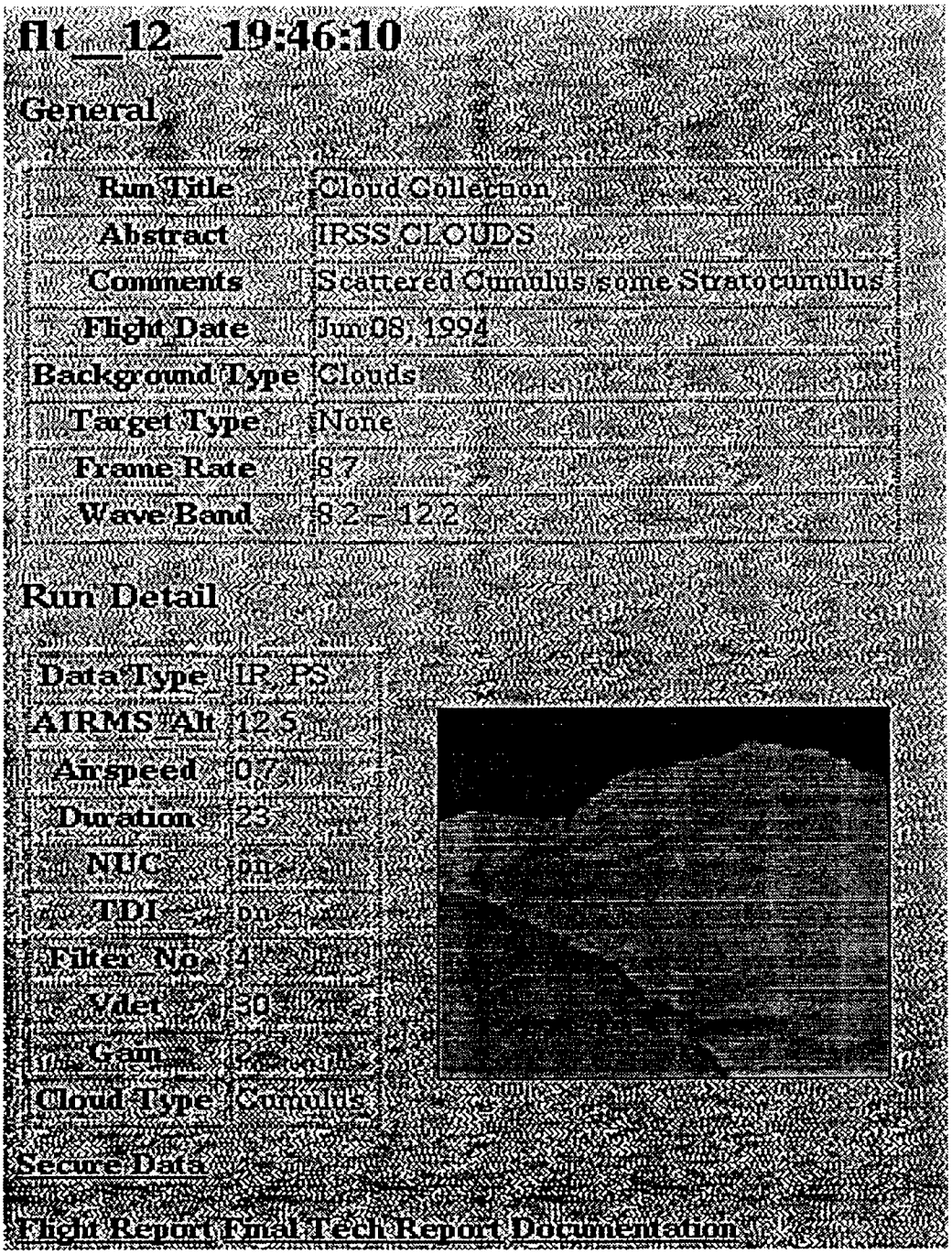


Figure 6. Detailed Database Output for Flight 12 including sample imagery

In addition to viewing detailed database outputs, a summary of key database parameters can be displayed for many flights on a single screen. This ability allows quick view a variety of flights and collection parameters and then select ones for downloading or processing. The quick summary can also be used as an archive list to which data was processed. One key parameter output from the Quick Summary is the Tape number which is important when requesting data from the MHPCC. Figure 7 shows an example of a quick summary output. The summary was generated by selecting 8 flight runs from the left browser panel and clicking on the summary table button under *viewers*.

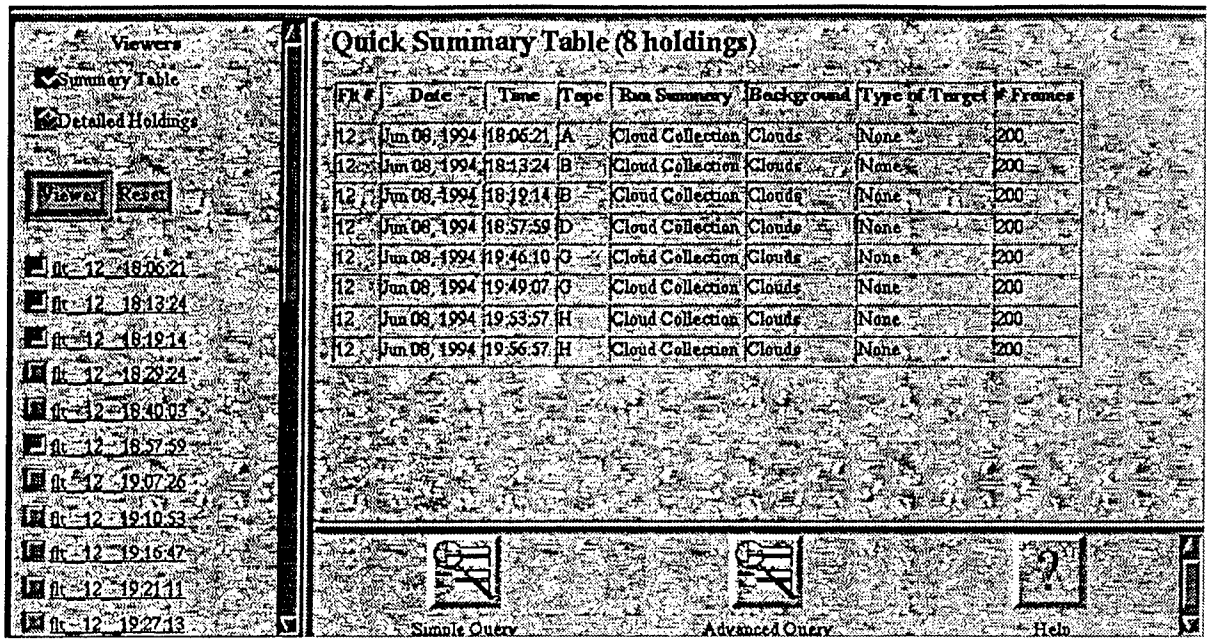


Figure 7. Quick Summary Database Output

3.1 Pre-Processing AIRMS Data

A state of the art end-to-end 3-D space-time processing stream was developed under the AIRMS program[1,2]. Portions of this distributed algorithm stream (DAS) have been ported to the MHPCC SP2 to provide a ready source of calibrated, "clean" and if desired, reworked data. After determining which AIRMS data to process, the user will be required to log into the MHPCC computer. Following the set up of Namelist files, the user can then submit the job. The user will be notified after the job completes, and can then download raw data and results. The AIRMS pre-processing algorithms include Data Quality Assessment, Calibration, Target Injection, Pattern Noise Mitigation, and Data Rework;

- *Data Quality Assessment* includes a number of checks to determine the quality of the data collected during a flight. These checks measure sensor noise, calibration, pointing accuracy, stabilization, and focusing.
- *Calibration* is an adjunct function to the two-point linear non-uniformity correction (NUC) performed within the sensor hardware. Experience has shown the two-point NUC to be adequate for most applications; however, for special applications, post-flight linear or quadratic recalibration has proven effective in reducing sensor artifacts, such as pattern noise, even further. The recalibration process can convert the AIRMS data from sensor A/D counts to radiometric values (radiances).
- *Target Injection* allows inserting additive point source (*i.e.*, fractional pixel) targets. Grids of targets with random velocities are typically inserted.
- *Pattern Noise Mitigation* attenuates row-to-row biases, as well as underlying 5-row and 25-row pattern noise (due to FPA readout hardware characteristics) present in the data after Calibration and/or NUC. Pattern noise is further reduced through application of an FIR spatial filter to the imagery

  1) *MUX-to-MUX offset removal*; implements level matching at the MUX boundaries;
  2) *Row-to-row bias removal*; implements bias removal linear filter; and
  3) *Five-line noise removal*; implements five-line noise removal linear filter.

- *Data Rework* provides the capability to transform AIRMS imagery so that it closely matches imagery that would be collected by a different sensor. The resolution, pattern noise, and NER of the AIRMS imagery put it at the forefront of today's IR sensors. Imagery from smaller aperture sensors can be simulated by reworking the AIRMS data. Reworking includes modifying the MTF, resampling, adding system and pattern noise, and adding intra- and inter-frame jitter.

Figures 8 and 9 illustrate the result of pre-processing raw AIRMS data through Data Quality Assessment, Calibration, Target Injection and Pattern Noise Mitigation algorithms. Figure 8 shows raw Flight 12 AIRMS data which contains pattern noise artifacts. Figure 9 shows the same data after pre-processing. Following pattern noise mitigation, it is clearly seen in Figure 9 that the row-to-row bias and MUX-to-MUX offsets have been removed from the raw image.

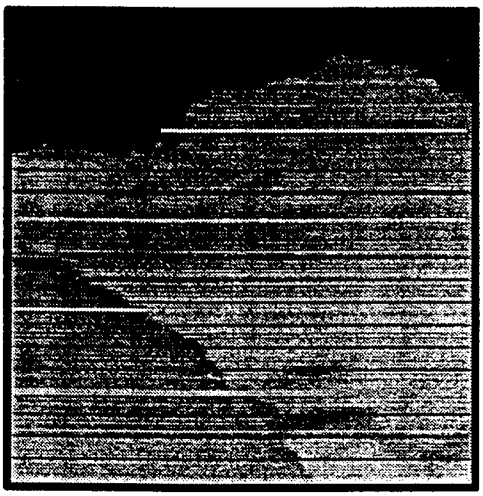


Figure 8. Raw Flight 12 Data with Pattern Noise

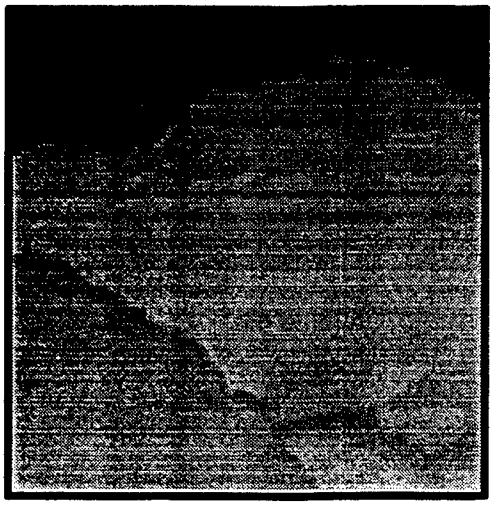


Figure 9. Processed Flight 12 Data Pattern Noise Removed

3.1.1 Download Data from the MHPCC for Local Processing

An example illustrating downloading data from the MHPCC to a user site shown below.

- The user links to the AIRMS Library World Wide Web (WWW) home page using Netscape or a similar WWW browser. This page, located on the MHPCC server, contains links to AIRMS background and reference material, an overview of the data library, an overview of the AIRMS algorithm stream, as well as a link to the library query page.
- The user peruses the data interactively using the query form provided. Searches of the data base can be requested by combining the various fields (*e.g.*, flight number, type of background, type of target, waveband, *etc.*) with logical operators (*e.g.*, and, or, =, >, <, *etc.*). The user also has the option to view MPEG images from the data files. In this way the user selects and identifies data sets relevant to his or her specific research.
- The user is given the option to download from the MHPCC all or any portion of the data selected. If the data is not currently on-line, the user will receive an immediate message. Once the data is placed on-line at the MHPCC, the user will be notified via e-mail. When the download function is initiated, the user will be asked to input a password. Password protection is the way used to ensure that only approved users can download full resolution image data.

The user then initiates any processing desired on his or her own workstation. Local processing software might include the serial version of the DAS, off-the-shelf packages such as Khoros, MATLAB, PV Wave; or custom software developed by the user.

3.1.2 Remotely Process Data at the MHPCC

An example illustrating the user interaction for remote data processing at the MHPCC is shown below.

- The user logs into the MHPCC through the Internet (*i.e.*, does a remote login from an open window on his or her workstation). Password protection is used to ensure that the user is approved to use the MHPCC.

- The user selects data using either the WWW query capability (identically to the previous example) or the more sophisticated AIRMS Librarian (a direct X window client-server application). A data selection previously saved by the user could also be recalled.
- The user interactively sets up algorithm and/or analysis runs, and identifies the AIRMS data to be processed. Packages are available to users include: the AIRMS pre-processing software and a general purpose image processing package. AIRMS uses text files to specify the parameters and processing options. These text files can be created on either the MHPCC or the user's local workstation.
- The user submits one or more processing runs. The actual processing of AIRMS data is then performed at the MHPCC and the user notified of completion by email.
- The user logs in again and either inspects the results at the MHPCC in a client server relationship with the user's workstation, or the user can download the results and inspect on his or her workstation directly with any analysis packages available locally.

4. SUMMARY

This paper has presented a top level description of the AIRMS Web-based IR Data library and pre-processing capabilities. The frame-based WWW interface provides users with a simple and efficient means of accessing, pre-processing and downloading AIRMS data. Users can query the AIRMS database using a multitude of search parameters including background, target type, number of frames, *ect.* Users can also preview sample AIRMS imagery data stored in MPEG format. A quick summary database output is available to allow users to quickly display summary information for many flights together in a single window. Expanded database detail information and a vast amount of documentation consisting of reports and manuals is also available on-line.

The intent of this paper is to generate high visibility and maximum utility for AIRMS data by providing access to the data and processing algorithms via the internet. This transfer of technology to the community takes advantage of existing government and contractor expertise, software, and facilities and provides the highest quality IR data in existence to the research community. In addition, access to a high performance computing facility (MHPCC IBM SP2) for data processing provides system/algorithm developers, physicists, etc., the ability to address a wide range of problems using the internet as well as traditional methods to acquire and process IR data.

ACKNOWLEDGMENTS

The authors greatly acknowledge the many contributors to the developers of the MHPCC program including Peter Wiley of the Naval Air Weapons Center, China Lake, and D.J. Fabozzi of the MHPCC.

REFERENCES

1. K. Ralston, J. Attili, et. al., *"The Airborne Infrared Measurement System Final Technical Report"*, PAR Government Systems Corporation, La Jolla, 1996.
2. Joseph B. Attili, Robert W. Fries, et. al., *"False Track Discrimination in a 3-D Signal/Track Processor"*, SPIE Proceedings (Signal and Data Processing of Small Targets Conference), 1996.

Bibliography/References

1. Tanenbaum, A.S,"Computer Networks", Prentice Hall, Englewood Cliffs, New Jersey, p429.
2. Postel, J. and Reynolds, J., "File Transfer Protocol (FTP), RFC 959", USC ISI, October 1988.
3. Krol, E. "The Whole Internet", O'Reilly & Associates, Inc, Sebastopol, CA.
4. Iannucci, D.J., Lekashman, J., "MFTP: Virtual TCP windo scaling using multiple connections".
5. "FTP manual pages", SunOS Release 4.1, January 1988.
6. The MathWorks Inc, MATLAB Compiler User's Guide, ***Publisher*** 1995.
7. V. Menon, A. Trefethen, "MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing", SuperComputing '97 Technical Paper.
8. Real Time Express, http://www.rtexpress.com/.
9. Luiz De Rose and David Padua, A MATLAB to Fortran 90 Translator and its Effectiveness, 10th ACM International Conference on Supercomputing, May 1996.
10. The RTExpress distribution example codes.
11. The Ground Processing Space Time Adaptive Processing (STAP) Suite courtesy of Massachusettes Institute of Technology/Loncoln Laboratory.
12. P.L. Springer, Matpar: Parallel Extensions for MATLAB, http://www.hpc.jpl.nasa.gov/PS/MATPAR/index.html.
13. Joel Hollingsworth, Kun Liu, and Paul Pauca, Parallel Toolbox for MATLAB, Wake Forest University, http://www.mthcfc.wfu.edu/pt/pt.html.
14. H. Anton, Elementary Linear Algebra, 1984 John Wiley and Sons, p69-86.
15. MATCOM V2, A MATLAB to C++ compiler Users Manual.
16. The Math Works, Inc. MATLAB Compiler, 1995.
17. The Math Works, Inc. MATLAB User's Guide, 1996.
18. The Math Works, Inc. MATLAB Reference Guide, 1996.
19. The Math Works, Inc. MATLAB External Interface Guide, 1996.
20. The Math Works, Inc. Corporate web pages located at http://www.mathworks.com.
21. Luiz De Rose and David Padua. A MATLAB to Fortran 90 Translator and its Effectiveness, March 1996.
22. V. Menon and A.E. Trefethen MultiMATLAB: Integrating MATLAB with High-performance Parallel Computing. Cornell Theory Center. http://simon.cs.cornell.edu/Info/People/usm/papers/sc97/
23. A.E. Trefethen, V.S. Menon, C.C. Chang, G.J. Czajkowski, C. Myers and L.N. Trefethen MultiMATLAB: MATLAB on multiple processors. Technical Report pp96-239, Cornell Theory Center, 1996. http://www.cs.cornell.edu/Info/People/Int/multimatlab.html

24. The Math Works Inc., MATLAB User's Guide: reference Guide, Compiler Guide. The Math Works Inc. 1992.

25. R. Butler and E. Lusk, Monitors, message and clusters: "The P4 Parallel Programming System", Parallel Computing, April, 1994.

26. MPICH-A Portable Implementation of MPI http://www.mcs.anl.gov/mpi/mpich/

27. C.C. Chang, G.J. Czajkowski, X. Liu, V.X. Menon, C. Myers A.E. Trefethen, and L.N. Trefethen. The Cornell MultiMATLAB Project. Cornell Theory Center. http://www.tc.cornell.edu/Software/MultiMATLAB/

28. The Math Works Technical Papers. http://www.mathworks.com/support/tech-notes/v5/1600/1615.shtml

29. The Math Works Inc. MATLAB User's Guide: Reference Guide. The Math Works Inc. 1992.

30. Integrated Sensors, Inc. RTExpress Real Time User's Guide, July 1998.

31. Integrated Sensors, Inc. Product web pages located at http://www.rtexpress.com.

32. K. Ralston, J. Attili, et. al., "The Airborne Infrared Measurement System Final Technical Report", PAR Government Systems Corporation, La Jolla, 1996.

33. Joseph B. Attili, Robert W. Fries, et. al., "False Track Discrimination in a 3-D Signal/Track Processor", SPIE Proceedings (Signal and Data Processing of Small Targets Conference), 1996.

*U.S. GOVERNMENT PRINTING OFFICE: 2001-710-038-10169